

**WHAT IF I TOLD YOU**



**NETWORK TOPOLOGIES ARE NO  
LONGER STATIC?**

generator.net

# Reconfigurable Networks: Enablers, Algorithms, Complexity

Ramakrishnan Durairajan, Klaus-Tycho Forster, Stefan Schmid

Tutorial @ ACM Sigmetrics 2019  
Phoenix, Arizona, USA

# Roadmap for this tutorial

- Reconfigurable Networks in three parts
  - Wide area networks
  - Data centers
  - Metrics
- 10 minutes break
- Interactive with Q&A
- Future outlook
  - Big open problems



---

# Part 1: Reconfigurable WANs

---

# Roadmap for reconfigurable WANs

- Reconfigurable Networks: what happens in a WAN?
  - Introduction to optical layer, traffic engineering, and capacity planning
- Optical layer support for dynamic capacity planning
- Future outlook
  - Challenges in realizing Reconfigurable WANs
  - Open problems in systems research



---

# Terminologies

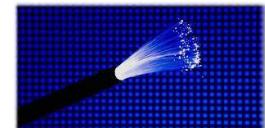
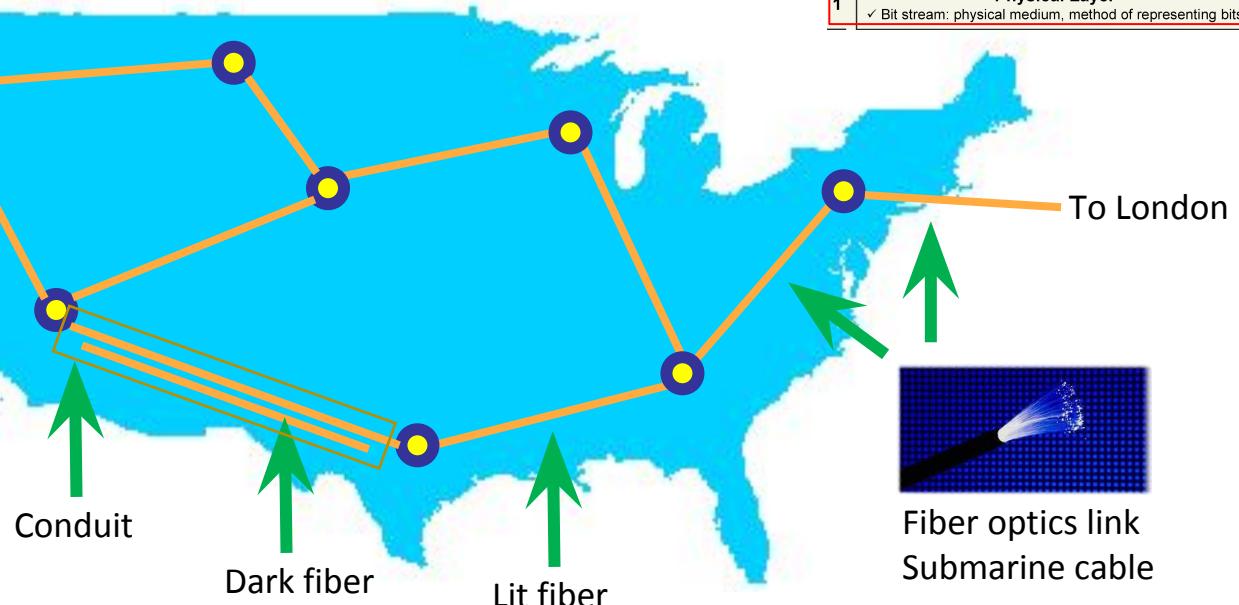
---

# Physical Internet



Point of Presence (POP)  
Internet Exchange Point (IXP)  
Datacenters (DCs)

...

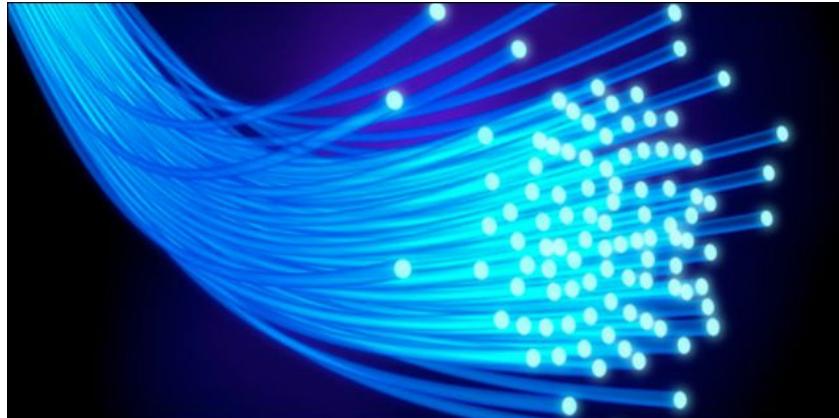


Fiber optics link  
Submarine cable

|   |   |
|---|---|
| 7 | <b>Application Layer</b><br>✓ Message format, Human-Machine Interfaces              |
| 6 | <b>Presentation Layer</b><br>✓ Coding into 1s and 0s; encryption, compression       |
| 5 | <b>Session Layer</b><br>✓ Authentication, permissions, session restoration          |
| 4 | <b>Transport Layer</b><br>✓ End-to-end error control                                |
| 3 | <b>Network Layer</b><br>✓ Network addressing; routing or switching                  |
| 2 | <b>Data Link Layer</b><br>✓ Error detection, flow control on physical link          |
| 1 | <b>Physical Layer</b><br>✓ Bit stream: physical medium, method of representing bits |

# Optical Layer

- Optical components in the physical layer
  - Optical fiber cables
    - In WANs, fibers are the freeways (100s of miles, cities)
    - Used as means to transmit light between endpoints



# Optical Layer

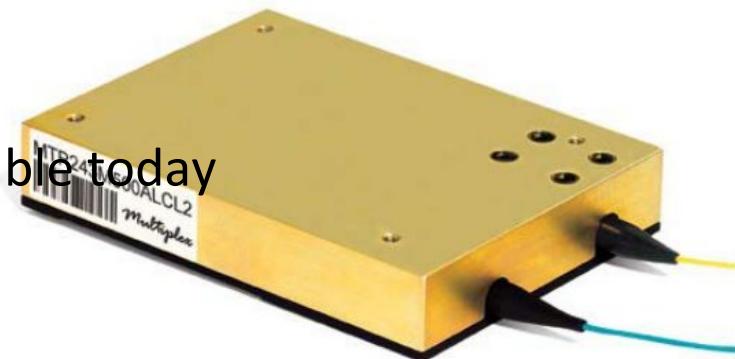
- Optical components in the physical layer
  - Optical fiber cables
  - Wavelengths (lights)
    - Independent channel of information

# Optical Layer

- Optical components in the physical layer
  - Optical fiber cables
  - Wavelengths (lights)
  - Dense Wavelength Division Multiplexing (DWDM)
    - O(100) of separate wavelengths can be multiplexed into a lightstream transmitted on a single optical fiber

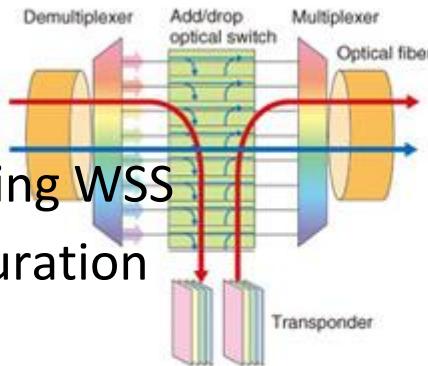
# Optical Layer

- Optical components in the physical layer
  - Optical fiber cables
  - Wavelengths (lights)
  - Dense Wavelength Division Multiplexing (DWDM)
  - Transponder
    - Optical -- Electrical conversion
    - Data rate is based on the modulation
      - 100 Gbps per wavelength is available today

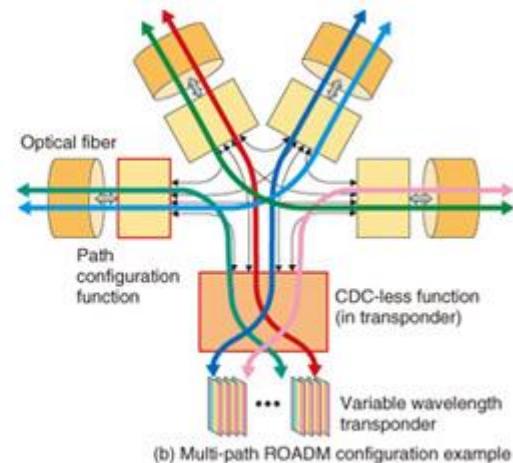


# Optical Layer

- Optical components in the physical layer
  - Optical fiber cables
  - Wavelengths (lights)
  - Dense Wavelength Division Multiplexing (DWDM)
  - Transponder
  - ROADM
    - Combines wavelengths using WSS
    - Enables dynamic reconfiguration



(a) Two-path ROADM configuration example



(b) Multi-path ROADM configuration example

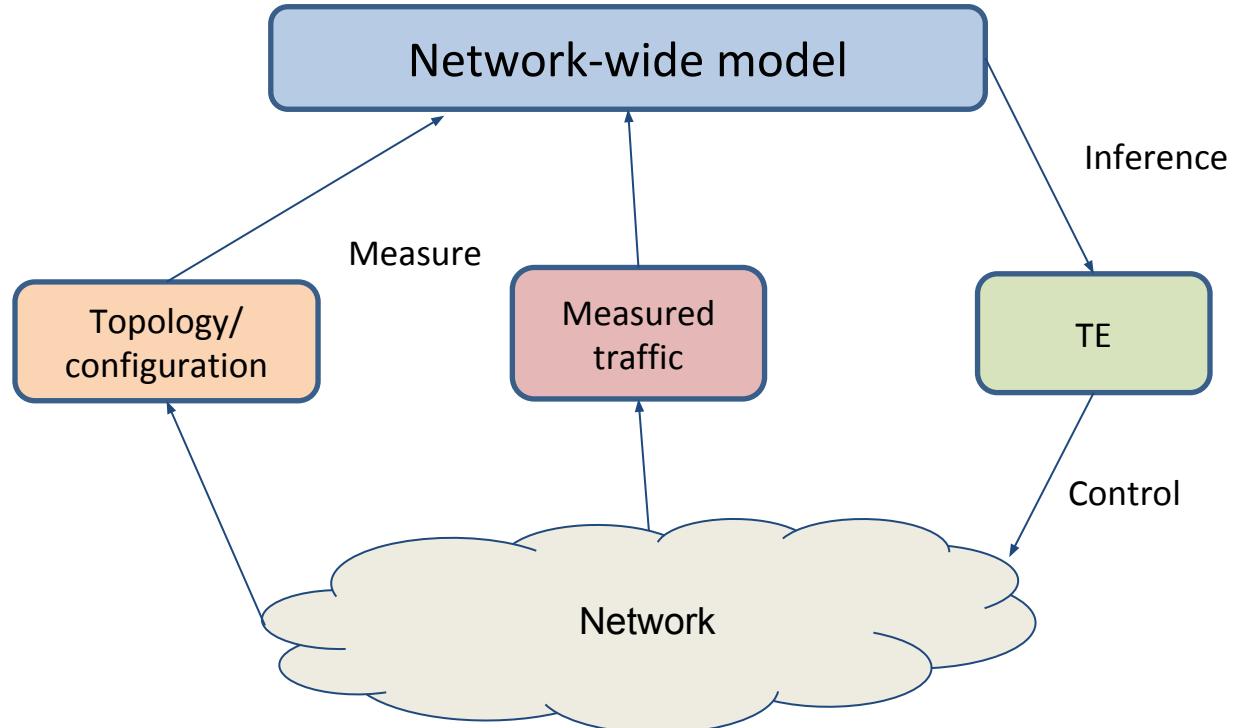
# Traffic Engineering

- Does the network run efficiently?
  - Are there congested links when there are idle links?
  - High-delay links when there are low-delay links?
- How should routing adapt to traffic?
  - How to avoid congested links?
  - What about topology changes? Failures?
- Long- and short-term capacity planning?
  - How to satisfy application requirements (e.g., delay)?
  - How to accommodate demands and peak traffic?

# Putting it all together

TE and Optical layers:

*“Two disconnected worlds!”*

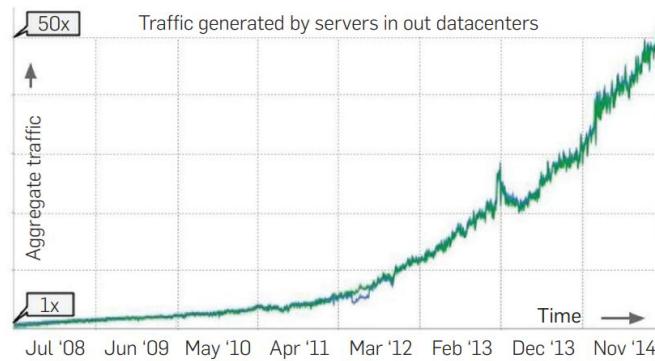


# Trend: Explosive growth of data!



# Explosive Growth of Demand...

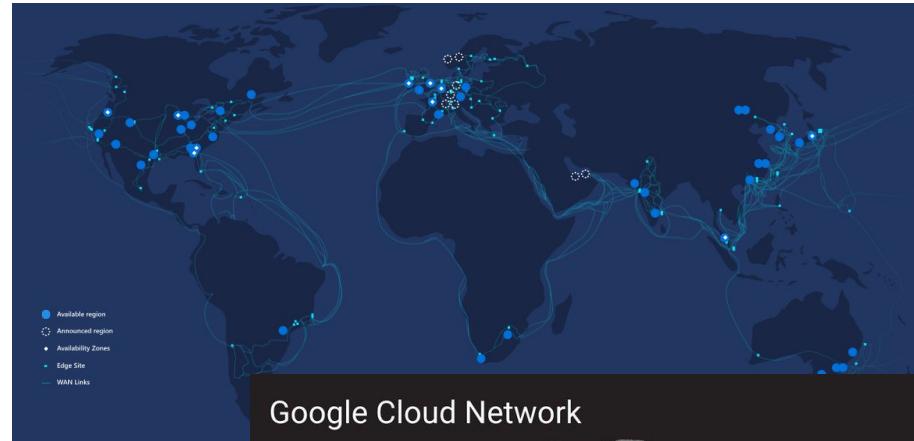
Batch processing, web services,  
**distributed ML**, ....: ***data-centric applications*** are distributed and  
interconnecting network is ***critical***



Source: Jupiter Rising. SIGCOMM 2015.

Aggregate server traffic in  
**Google's datacenter fleet**

# ... Leads to Private WANS!



Google Cloud Network



# Battle of the Global Backbones:

What are Your Options?

Cloud  
Networks?

The Public  
Internet?

Global MPLS  
Services?



# Desiderata



Maximize utilization



Minimize costs



Tolerate failures

# Challenges

- Optical backbones are expensive
  - Billions of dollars
- Optical backbones are inefficient
  - Unknown traffic demands
    - Periodically provisioned to accommodate peak traffic
  - Unknown failure scenarios
    - Built-in margins to accommodate failures

# Reality



Maximize utilization



Overprovisioned (2x the avg.)



Minimize costs



Tolerate failures

# Reality



Maximize utilization



Overprovisioned (2x the avg.)



Minimize costs



Costs billions of dollars



Tolerate failures

# Reality



Maximize utilization



Overprovisioned (2x the avg.)



Minimize costs



Costs billions of dollars



Tolerate failures



Built-in failure margins

# State-of-the-art

- OWAN
- SD-OWAN
- RADWAN
- and many more...

In a nutshell: optical layer is static!

---

What if we can move beyond these solutions?

---

# Dynamic Capacity Planning

- Wavelength allocations are no longer fixed or static
- Wavelengths are reconfigured in response to changing traffic demands and unexpected failure scenarios

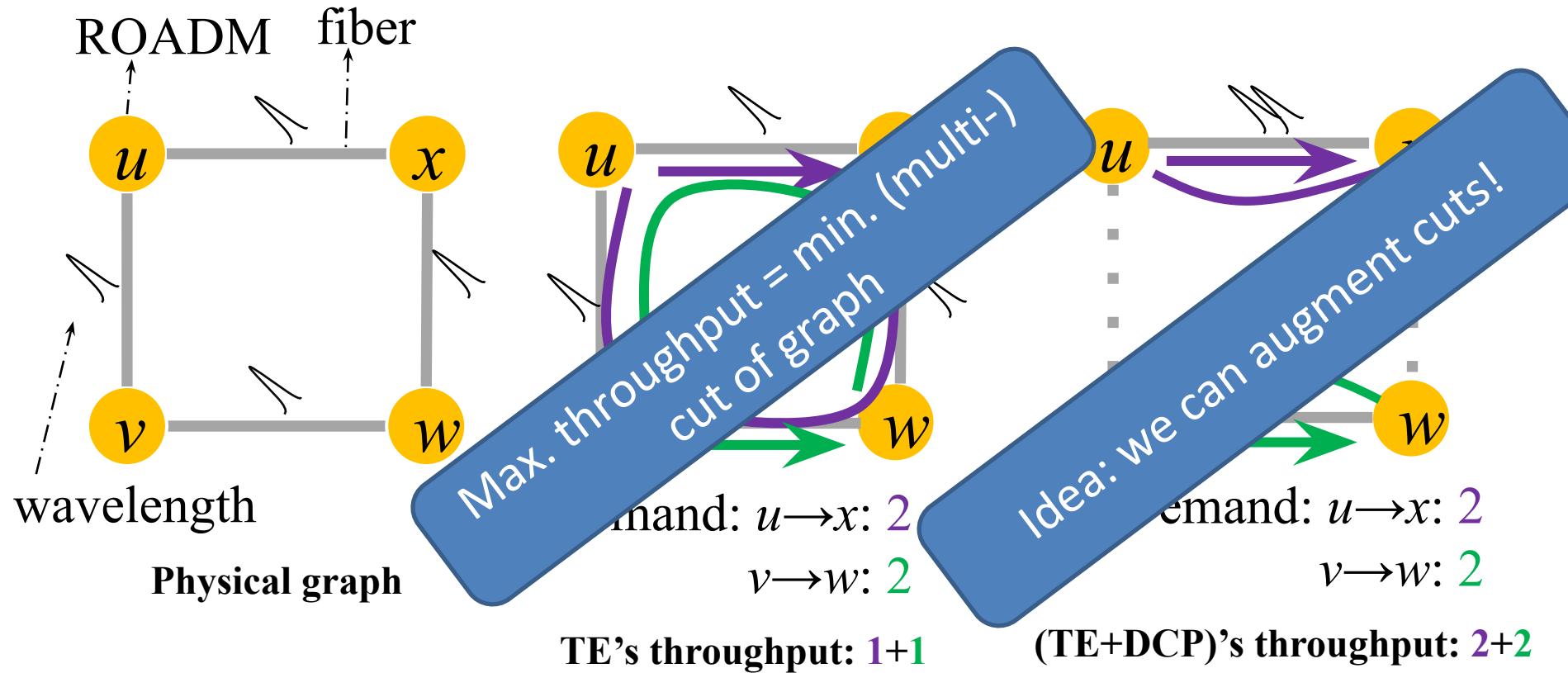
Goal: best utilize the deployed fiber  
*without SDN!*

---

What are the exact scenarios and potential gains of DCP at the optical layer?

---

# Throughput Gains



# Opportunity 1



## Spatial traffic shift

Without DCP, providers over-provision the WAN based on worst-case traffic predictions

Requires fast reconfiguration time in the optical domain!

# Opportunity 2



## Failure induced traffic shift

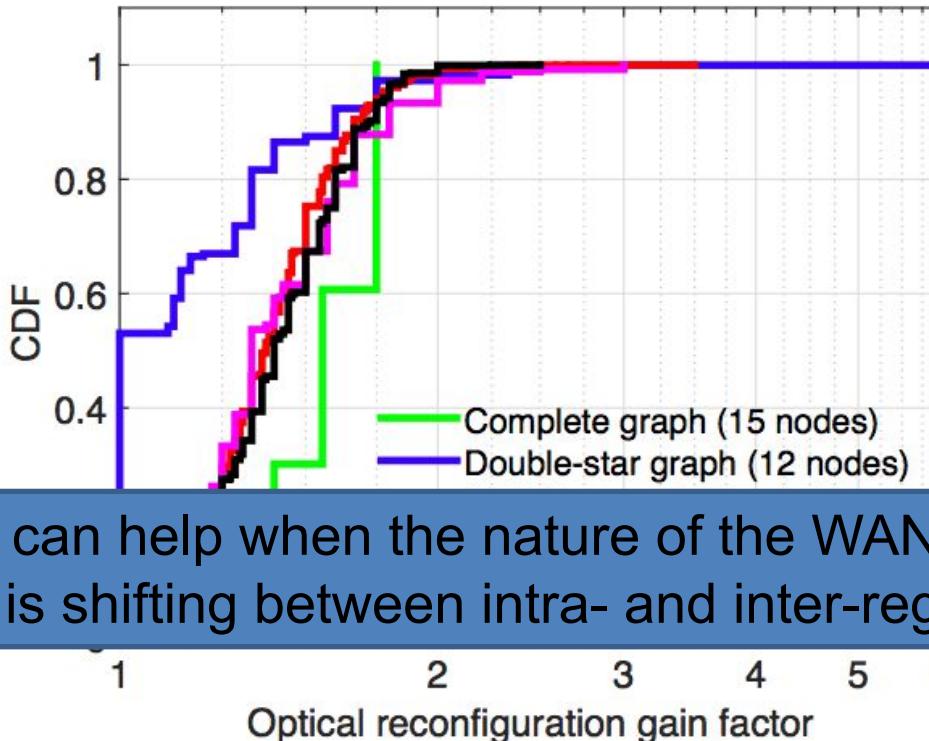
Without DCP, providers would need to account for additional wavelengths as stand-by resources

*Pro-actively* reconfigures wavelengths on backup paths without impacting TE

# Gain on Real Topologies

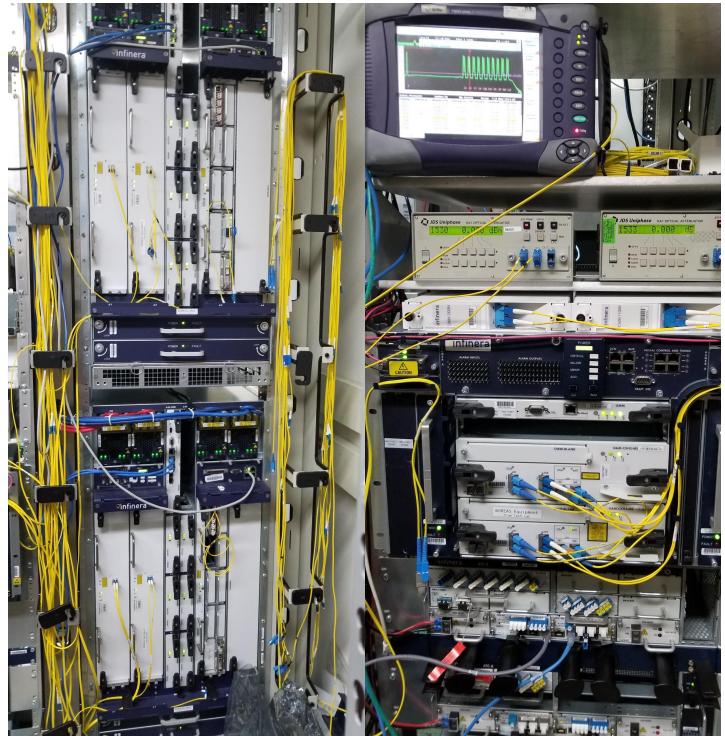
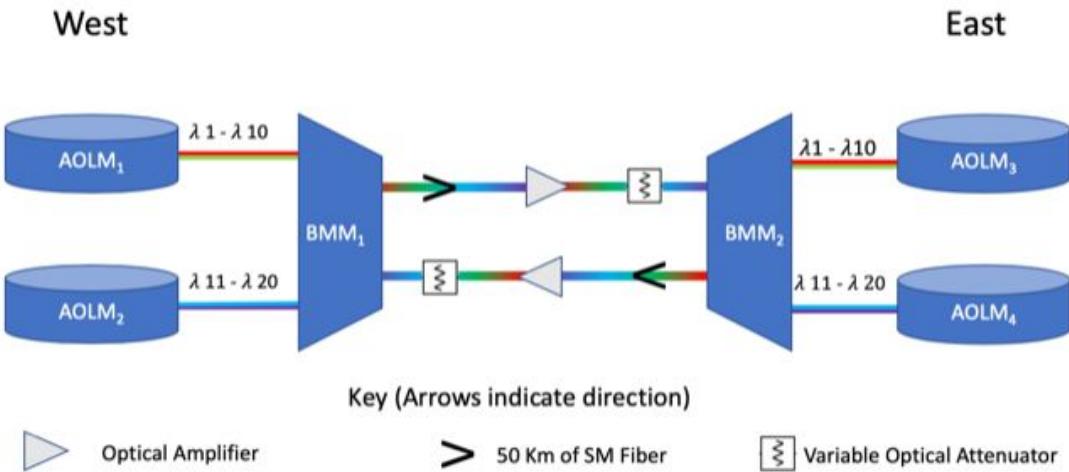
- We consider five topologies
  - B4, SWAN, Internet2, Complete, Double-star
- Integer min-cost flow problem with a virtual super-source/destination
- Obtain polynomial runtime for every partition computation

# Gain on Real Topologies



DCP can help when the nature of the WAN traffic demand is shifting between intra- and inter-region traffic.

# Lab-based Experiments



# Lab-based Experiments

- We measured the worst case for power stabilization
  - Default mode: ~10 minutes
  - Manual mode\* (without safeguards): ~4 seconds

Time required for light paths to stabilize after changes  
can be achieved as fast as 4s!

\* Turns AGC off. AGC is responsible for maintaining a suitable signal amplitude at its output, despite variation of the signal amplitude at the input.

Given a **topology** and a distribution of **edge failures**,

---

# How to allocate backup wavelengths on network paths?

---

Minimizing **cost** and ensuring **fault tolerance?**

# Overview

CapEx = amortized cost of new optical networking equipment to meet expected demands

OpEx = cost of “lost capacity” due to the time required to reconfigure light paths in the network in response to an edge failure

If  $\text{CapEx} > \text{OpEx}$ , service provider has no gains for deploying backup paths!

# Formulation

$$T_{cap}/L * E \sum_{p \in P} C_v + \sum_{w \in W} (\prod_{l \in w} p_l \sum_{v \in V_w} T_{op} * R * Y_{v,w})$$

Cost of transponder equipment in dollars/Gbps

$$\forall (i, j) \in \{D | i \neq j\} \text{ } \& \text{ } \forall w \in W : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v + \sum_{v \in V_w | Z(p, i, j)} Y_{v,w} \geq D_{i,j}$$

$$\forall p \in P \text{ } \& \text{ } w \in W : \\ Y_{v,w} \leq C_v$$

$$\forall (i, j) \in \{D | i \neq j\} : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v \geq D_{i,j}$$

$$\forall v \in V : \\ C_v \geq D_{i,j} / n_{i,j}$$

# Formulation

$$T_{cap} / L * E \sum_{p \in P} C_v + \sum_{w \in W} (\prod_{l \in w} p_l \sum_{v \in V_w} T_{op} * R * Y_{v,w})$$

Equipment lifetime in years

$$\forall (i, j) \in \{D | i \neq j\} \text{ & } \forall w \in W : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v + \sum_{v \in V_w | Z(p, i, j)} Y_{v,w} \geq D_{i,j}$$

$$\forall p \in P \text{ & } w \in W : \\ Y_{v,w} \leq C_v$$

$$\forall (i, j) \in \{D | i \neq j\} : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v \geq D_{i,j}$$

$$\forall v \in V : \\ C_v \geq D_{i,j} / n_{i,j}$$

# Formulation

$$T_{cap}/L * E \sum_{p \in P} C_v + \sum_{w \in W} (\prod_{l \in w} p_l \sum_{v \in V_w} T_{op} * R * Y_{v,w})$$

Epoch at which the optimization is evaluated

$$\forall (i, j) \in \{D | i \neq j\} \text{ & } \forall w \in W : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v + \sum_{v \in V_w | Z(p, i, j)} Y_{v,w} \geq D_{i,j}$$

$$\forall p \in P \text{ & } w \in W : \\ Y_{v,w} \leq C_v$$

$$\forall (i, j) \in \{D | i \neq j\} : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v \geq D_{i,j}$$

$$\forall v \in V : \\ C_v \geq D_{i,j} / n_{i,j}$$

# Formulation

$$T_{cap}/L * E \sum_{p \in P} C_v - \sum_{w \in W} (\prod_{l \in w} p_l \sum_{v \in V_w} T_{op} * R * Y_{v,w})$$

Capacity to allocate for path "v"

$$\forall (i, j) \in \{D | i \neq j\} \text{ & } \forall w \in W : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v + \sum_{v \in V_w | Z(p, i, j)} Y_{v,w} \geq D_{i,j}$$

$$\forall p \in P \text{ & } w \in W : \\ Y_{v,w} \leq C_v$$

$$\forall (i, j) \in \{D | i \neq j\} : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v \geq D_{i,j}$$

$$\forall v \in V : \\ C_v \geq D_{i,j} / n_{i,j}$$

# Formulation

$$T_{cap}/L * E \sum_{p \in P} C_v + \sum_{w \in W} \left( \prod_{l \in w} p_l \right) \sum_{v \in V_w} T_{op} * R * Y_{v,w}$$

Probability of link "l" failing

$$\forall (i, j) \in \{D | i \neq j\} \text{ & } \forall w \in W : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v + \sum_{v \in V_w | Z(p, i, j)} Y_{v,w} \geq D_{i,j}$$

$$\forall p \in P \text{ & } w \in W : \\ Y_{v,w} \leq C_v$$

$$\forall (i, j) \in \{D | i \neq j\} : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v \geq D_{i,j}$$

$$\forall v \in V : \\ C_v \geq D_{i,j} / n_{i,j}$$

# Formulation

$$T_{cap}/L * E \sum_{p \in P} C_v + \sum_{w \in W} \left( \prod_{l \in w} p_l \sum_{v \in V_w} T_{op} * R * Y_{v,w} \right)$$

Set of fiber failure/cut scenarios

$$\forall (i,j) \in \{D | i \neq j\} \text{ & } \forall w \in W : \\ \sum_{v \notin V_w | Z(p,i,j)} C_v + \sum_{v \in V_w | Z(p,i,j)} Y_{v,w} \geq D_{i,j}$$

$$\forall p \in P \text{ & } w \in W : \\ Y_{v,w} \leq C_v$$

$$\forall (i,j) \in \{D | i \neq j\} : \\ \sum_{v \notin V_w | Z(p,i,j)} C_v \geq D_{i,j}$$

$$\forall v \in V : \\ C_v \geq D_{i,j} / n_{i,j}$$

# Formulation

$$T_{cap}/L * E \sum_{p \in P} C_v + \sum_{w \in W} (\prod_{l \in w} p_l \sum_{v \in V_w} T_{op} * R * Y_{v,w})$$

Cost of transmitting a Gb of data

$$\forall (i, j) \in \{D | i \neq j\} \text{ & } \forall w \in W : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v + \sum_{v \in V_w | Z(p, i, j)} Y_{v,w} \geq D_{i,j}$$

$$\forall p \in P \text{ & } w \in W : \\ Y_{v,w} \leq C_v$$

$$\forall (i, j) \in \{D | i \neq j\} : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v \geq D_{i,j}$$

$$\forall v \in V : \\ C_v \geq D_{i,j} / n_{i,j}$$

# Formulation

$$T_{cap}/L * E \sum_{p \in P} C_v + \sum_{w \in W} (\prod_{l \in w} p_l \sum_{v \in V_w} T_{op} * R * Y_{v,w})$$

Reconfiguration time

$$\forall (i, j) \in \{D | i \neq j\} \text{ } \& \text{ } \forall w \in W : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v + \sum_{v \in V_w | Z(p, i, j)} Y_{v,w} \geq D_{i,j}$$

$$\forall p \in P \text{ } \& \text{ } w \in W : \\ Y_{v,w} \leq C_v$$

$$\forall (i, j) \in \{D | i \neq j\} : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v \geq D_{i,j}$$

$$\forall v \in V : \\ C_v \geq D_{i,j} / n_{i,j}$$

# Formulation

$$T_{cap}/L * E \sum_{p \in P} C_v + \sum_{w \in W} \left( \prod_{l \in w} p_l \sum_{v \in V_w} T_{op} * R * Y_{v,w} \right)$$

Capacity to reconfigure on path "v" due to "w"

$$\forall (i, j) \in \{D | i \neq j\} \text{ } \& \text{ } \forall w \in W : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v + \sum_{v \in V_w | Z(p, i, j)} Y_{v,w} \geq D_{i,j}$$

$$\forall p \in P \text{ } \& \text{ } w \in W : \\ Y_{v,w} \leq C_v$$

$$\forall (i, j) \in \{D | i \neq j\} : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v \geq D_{i,j}$$

$$\forall v \in V : \\ C_v \geq D_{i,j} / n_{i,j}$$

# Formulation

$$T_{cap}/L * E \sum_{p \in P} C_v + \sum_{w \in W} \left( \prod_{l \in w} p_l \sum_{v \in V_w} T_{op} * R * Y_{v,w} \right)$$

$$\forall (i, j) \in \{D | i \neq j\} \text{ } \& \text{ } \forall w \in W : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v + \sum_{v \in V_w | Z(p, i, j)} Y_{v,w} \geq D_{i,j}$$

For all demands in the traffic matrix and all fiber cut scenarios, there must be enough capacity on the available paths given some amount of additional capacity (through now-unused transponders) that is reclaimed after the fiber-cut.

$$\forall p \in P \text{ } \& \text{ } w \in W : \\ Y_{v,w} \leq C_v$$

$$\forall (i, j) \in \{D | i \neq j\} : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v \geq D_{i,j}$$

$$\forall v \in V : \\ C_v \geq D_{i,j} / n_{i,j}$$

# Formulation

$$T_{cap}/L * E \sum_{p \in P} C_v + \sum_{w \in W} (\prod_{l \in w} p_l \sum_{v \in V_w} T_{op} * R * Y_{v,w})$$

$$\forall (i,j) \in \{D | i \neq j\} \text{ } \& \text{ } \forall w \in W : \\ \sum_{v \notin V_w | Z(p,i,j)} C_v + \sum_{v \in V_w | Z(p,i,j)} Y_{v,w} \geq D_{i,j}$$

The amount of capacity that can be moved to an alternate path during a fiber-cut ( $w$ ) is not greater than the original capacity on the path ( $v$ ) whose edge was cut.

$$\forall p \in P \text{ } \& \text{ } w \in W : \\ Y_{v,w} \leq C_v$$

$$\forall (i,j) \in \{D | i \neq j\} : \\ \sum_{v \notin V_w | Z(p,i,j)} C_v \geq D_{i,j}$$

$$\forall v \in V : \\ C_v \geq D_{i,j} / n_{i,j}$$

# Formulation

$$T_{cap}/L * E \sum_{p \in P} C_v + \sum_{w \in W} (\prod_{l \in w} p_l \sum_{v \in V_w} T_{op} * R * Y_{v,w})$$

$$\forall (i, j) \in \{D | i \neq j\} \text{ } \& \text{ } \forall w \in W : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v + \sum_{v \in V_w | Z(p, i, j)} Y_{v,w} \geq D_{i,j}$$

Sum of capacity allocated to a set of paths needs to be no less than the demand between the two ends of those paths.

$$\forall p \in P \text{ } \& \text{ } w \in W : \\ Y_{v,w} \leq C_v$$

$$\forall (i, j) \in \{D | i \neq j\} : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v \geq D_{i,j}$$

$$\forall v \in V : \\ C_v \geq D_{i,j} / n_{i,j}$$

# Formulation

$$T_{cap}/L * E \sum_{p \in P} C_v + \sum_{w \in W} (\prod_{l \in w} p_l \sum_{v \in V_w} T_{op} * R * Y_{v,w})$$

$$\forall (i, j) \in \{D | i \neq j\} \text{ } \& \text{ } \forall w \in W : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v + \sum_{v \in V_w | Z(p, i, j)} Y_{v,w} \geq D_{i,j}$$

Load-balancing constraint to ensure that the capacity allocated to a path is no less than an equitably split fraction of the demand on the path.

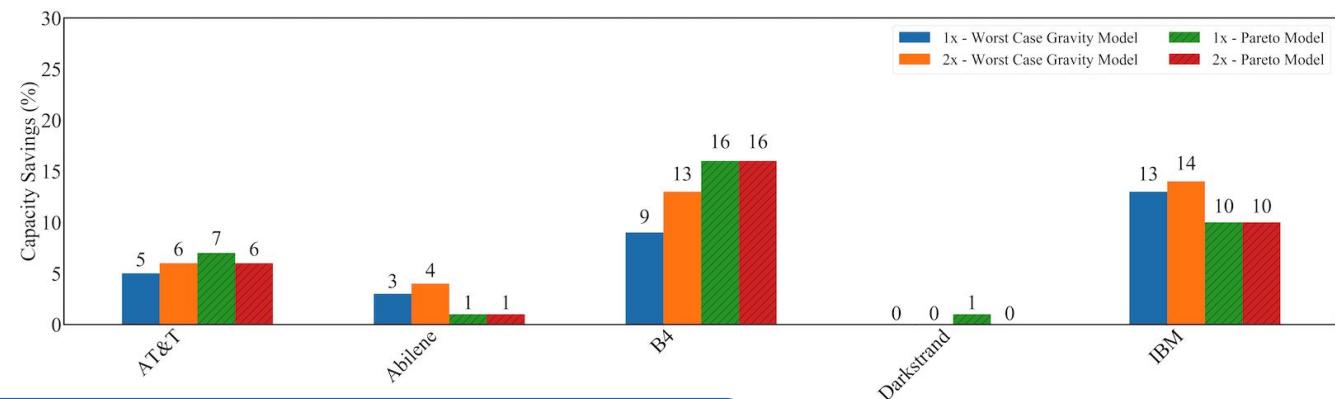
$$\forall p \in P \text{ } \& \text{ } w \in W : \\ Y_{v,w} \leq C_v$$

$$\forall (i, j) \in \{D | i \neq j\} : \\ \sum_{v \notin V_w | Z(p, i, j)} C_v \geq D_{i,j}$$

$$\forall v \in V : \\ C_v \geq D_{i,j} / n_{i,j}$$

# Capacity Savings

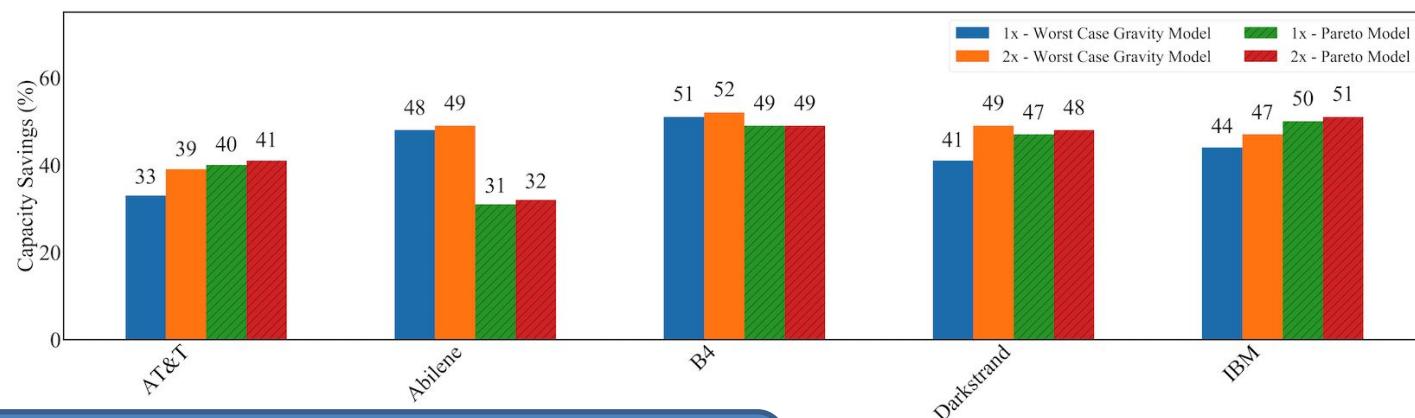
|             | AT&T | Abilene | B4 | Darkstrand | IBM |
|-------------|------|---------|----|------------|-----|
| Gravity     | 11   | 5       | 4  | 2          | 9   |
| Worst Case  | 5    | 3       | 9  | 0          | 13  |
| Exponential | 11   | 3       | 9  | 1          | 10  |
| Constant    | 11   | 5       | 11 | 1          | 12  |
| Pareto      | 7    | 1       | 16 | 1          | 10  |



Assumes edge failure probability less than 10% and reconfiguration time for a lambda of 3s.

# Capacity Savings

|             | AT&T | Abilene | B4 | Darkstrand | IBM |
|-------------|------|---------|----|------------|-----|
| Gravity     | 43   | 50      | 50 | 46         | 51  |
| Worst Case  | 33   | 48      | 51 | 41         | 44  |
| Exponential | 36   | 37      | 44 | 44         | 44  |
| Constant    | 43   | 47      | 52 | 49         | 50  |
| Pareto      | 40   | 31      | 49 | 47         | 50  |



Assumes edge failure probability less than 10% and reconfiguration time for a lambda of 100ms.

---

# Future Outlook

---

# Open problem 1

- SDN-like system to realize DCP
  - High performance, commercial-grade system
  - Leverage recent developments (e.g., OpenConfig)
  - Partnerships with service and equipment providers

# Open problem 2

- Testing capabilities and frameworks
  - Operational scenarios and network conditions
  - Consistency in the face of demands

# Open problem 3

- Physical layer-aware DCP
  - Link quality factor (q-factor) for better TE by Ghobadi et al.
  - Replace the failure probabilities with q-factor

# Open problem 4

- GreyLambda as a Service
  - Technology trends indicates feasibility!
  - Economic incentives for cloud providers?
  - Market trends?
  - How about “Fiber Exchange Points”?
    - What is the revenue model?
    - Cloud-like spot pricing?

# References

- Optimizing Bulk Transfers with Software-Defined Optical WAN. In SIGCOMM 2016.
- GreyLambda: Leveraging Optical Layer for Dynamic Capacity Planning. In submission.

# Acknowledgements

- Tutorial: Stefan Schmid and Klaus-Tycho Foerster
- Student: **Matthew Hall**
- Collaborators: Paul Barford, Mike Blodgett, Manya Ghobadi, William Jenson, Joel Sommers, Walter Willinger
- Images from Google

# Thank you!

Prof. Ram Durairajan, Ph.D.,  
Assistant Professor, CIS,  
Co-Director, Oregon Networking Research Group,  
University of Oregon

<https://ix.cs.uoregon.edu/~ram/>

# Backup

# Backup

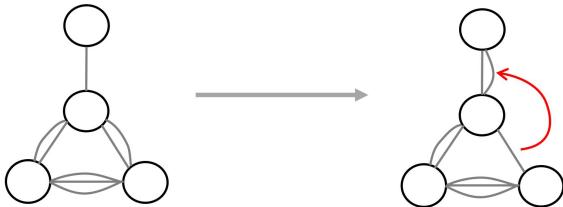
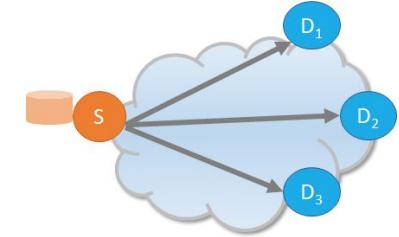
# Reconfigurable Networks: Enablers, Algorithms, Complexity

Ramakrishnan Durairajan, Klaus-Tycho Forster, Stefan Schmid

Tutorial @ ACM Sigmetrics 2019  
Phoenix, Arizona, USA

# DarTree: Reconfigurable WAN Multicast

- **Multicast transfers:** common in WAN
  - E.g., 90% multicast traffic of inter-DC traffic from **Baidu**

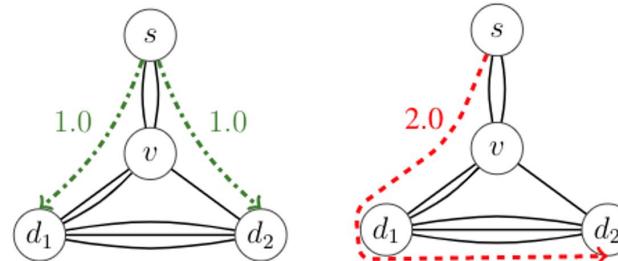


- How to support in reconfigurable WANs?
  - Can change **wavelengths** and **routing!**

Large potential and interesting optimization problem

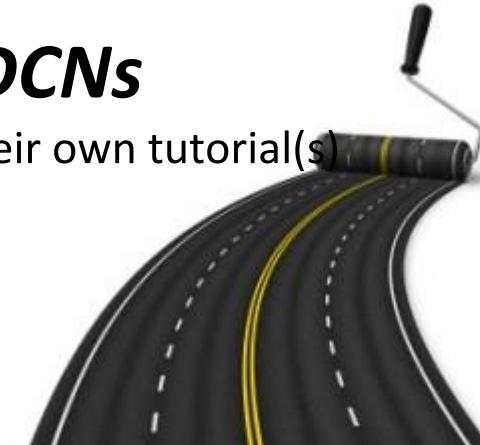
Just appeared at IWQoS 2019

<https://doi.org/10.1145/3326285.3329063>



# Roadmap

- Last part: WAN, now: Data Center Networks (DCNs)
  - This part: Mostly 1) estimate demand, 2) build topologies, 3) repeat
  - Next part: Dynamic settings, demand structures, data structure connections
- Focus on ***Algorithmic Challenges*** in ***DCNs***
  - Technology, system developments etc. would require their own tutorial(s)
- ***Paper-based*** approach
  - Selection of papers from 2009 to 2019



# Timeline

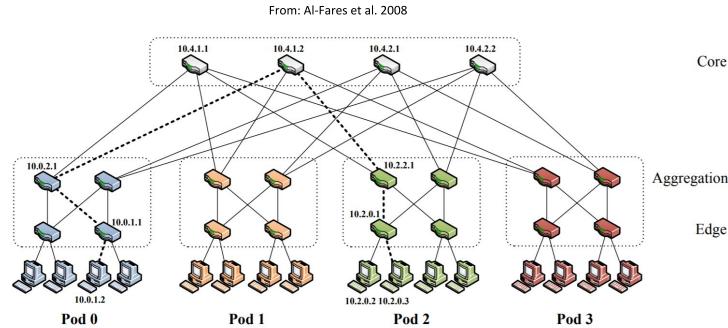
Reconfiguration time: from milliseconds **to microseconds** (and decentralized).

**Survey of Reconfigurable Data Center Networks.** Foerster and Schmid. SIGACT News, 2019.

- 2009
  - *Flyways* [51]: Steerable antennas (narrow beamwidth at 60 GHz [78]) to serve hotspots
- 2010
  - *Helios* [33]/*c-Through* [98, 99]: Hybrid switch architecture, maximum matching (Edmond's algorithm [30]), single-hop reconfigurable connections ( $O(10)ms$  reconfiguration time).
  - *Proteus* [21, 89]:  $k$  reconfigurable connections per ToR, multi-hop path stitching, multi-hop reconfigurable connections (weighted  $b$ -matching [69], edge-exchanges for connectivity [72], wavelength assignment via edge-coloring [67] on multigraphs)
- 2011
  - Extension of *Flyways* [51] to better handle practical concerns such as stability and interference for 60GHz links, along with greedy heuristics for dynamic link placement [45]
- 2012
  - *Mirror Mirror on the ceiling* [106]: 3D-beamforming (60 Ghz wireless), signals bounce off the ceiling
- 2013
  - *Mordia* [31, 32, 77]: Traffic matrix scheduling, matrix decomposition (Birkhoff-von-Neumann (BvN) [18, 97]), fiber ring structure with wavelengths ( $O(10)\mu s$  reconfiguration time)
  - *SplayNets* [6, 76, 82]: Fine-grained and online reconfigurations in the spirit of self-adjusting datastructures (all links are reconfigurable), aiming to strike a balance between short route lengths and reconfiguration costs
- 2014
  - *REACToR* [56]: Buffer burst of packets at end-hosts until circuit provisioned, employs [77]
  - *Firefly* [14] Combination of Free Space Optics and Galvo/switchable mirrors (small fan-out)
- 2015
  - *Solstice* [57]: Greedy perfect matching based hybrid scheduling heuristic that outperforms BvN [77]
  - Designs for optical switches with a reconfiguration latency of  $O(10)ns$  [3]
- 2016
  - *ProjecToR* [39]: Distributed Free Space Optics with digital micromirrors (high fan-out) [38] (Stable Matching [26]), goal of (starvation-free) low latency
  - *Eclipse* [95, 96]:  $(1 - 1/e^{(1-\varepsilon)})$ -approximation for throughput in traffic matrix scheduling (single-hop reconfigurable connections, hybrid switch architecture), outperforms heuristics in [57]
- 2017
  - *DAN* [7, 8, 11, 12]: Demand-aware networks based on reconfigurable links only and optimized for a demand snapshot, to minimized average route length and/or minimize load
  - *MegaSwitch* [23]: Non-blocking circuits over multiple fiber rings (stacking rings in [77] doesn't suffice)
  - *Rotornet* [63]: Oblivious cyclical reconfiguration w. selector switches [64] (Valiant load balancing [94])
  - *Tale of Two Topologies* [105]: Convert locally between Clos [24] topology and random graphs [87, 88]
- 2018
  - *DeepConf* [81]/*xWeaver* [102]: Machine learning approaches for topology reconfiguration
- 2019
  - Complexity classifications for weighted average path lengths in reconfigurable topologies [34, 35, 36]
  - *ReNet* [13] and *Push-Down-Trees* [9] providing statically and dynamically optimal reconfigurations
  - *DisSplayNets* [75]: fully decentralized *SplayNets*
  - *Opera* [60]: Maintaining expander-based topologies under (oblivious) reconfiguration

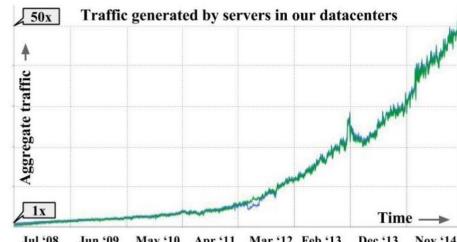
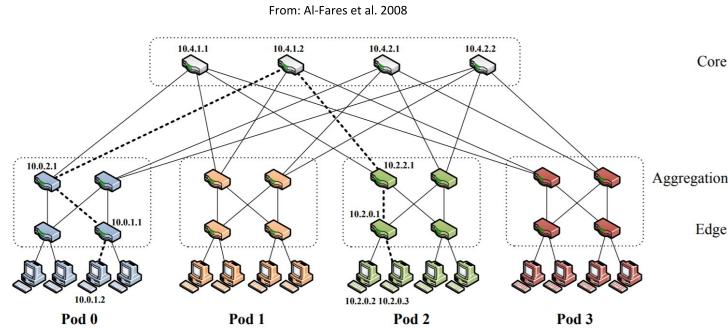
# Today's Data Center Topologies

- Often *Clos-based* (e.g. *Fat-tree*)
  - Goal: optimize for all-to-all communication
    - Idea: Obtain good bisection bandwidth



# Today's Data Center Topologies

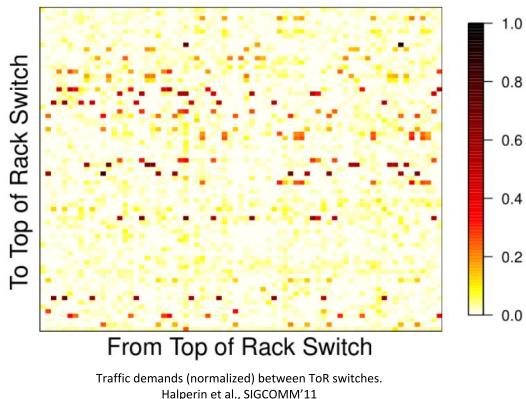
- Often *Clos-based* (e.g. *Fat-tree*)
  - Goal: optimize for all-to-all communication
    - Idea: Obtain good bisection bandwidth
- However, traffic is growing at unprecedented rates
  - What can we do?
  - Exponentially bigger networks?



From Google's Datacenter Network. Singh et al., SIGCOMM'15

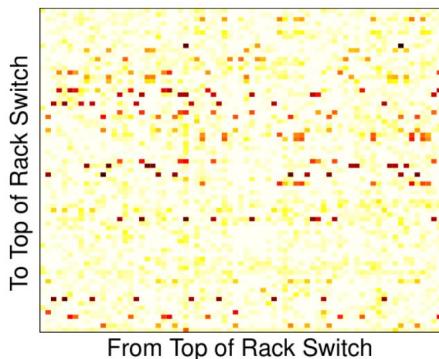
# Data Center Traffic $\neq$ Uniform

- However, DCN traffic is often **not** all-to-all

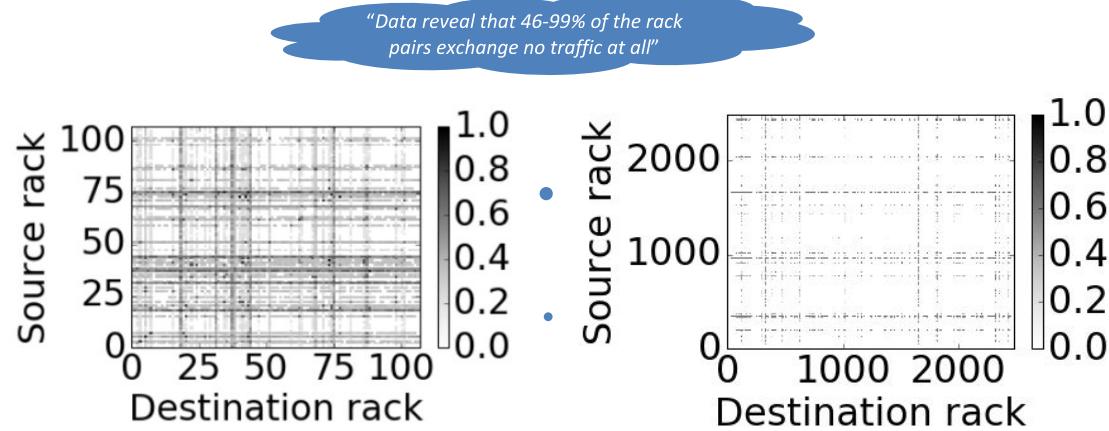


# Data Center Traffic $\neq$ Uniform

- However, DCN traffic is often **not** all-to-all



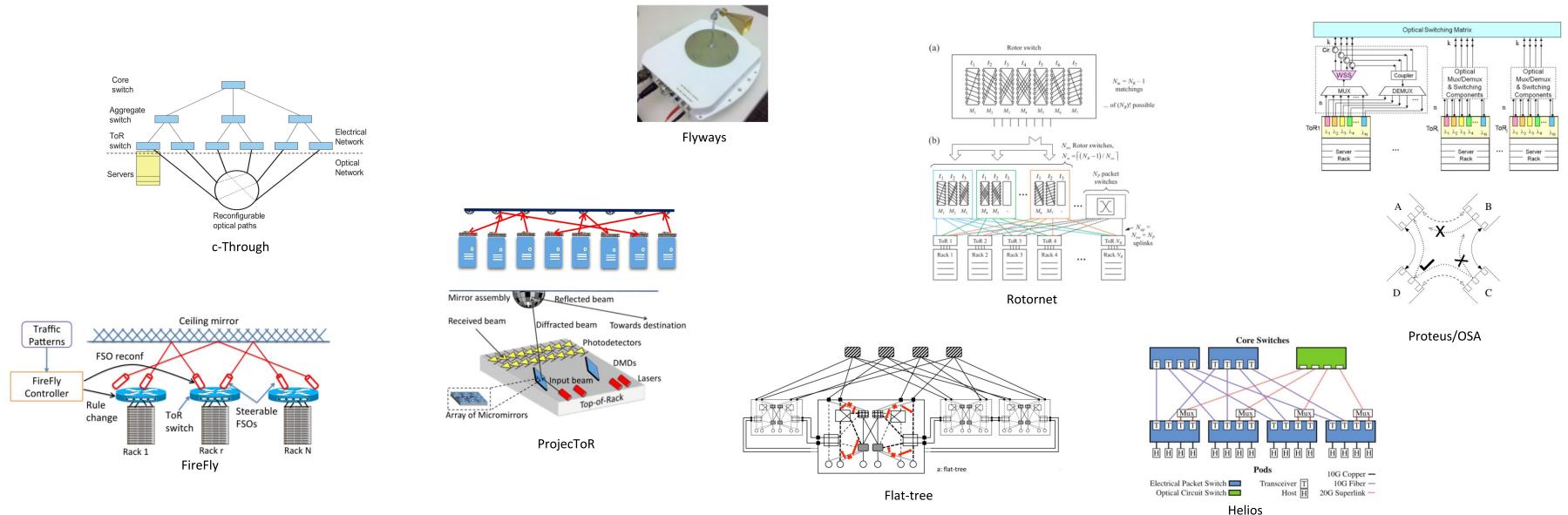
Traffic demands (normalized) between ToR switches.  
Halperin et al., SIGCOMM'11



Heatmap of rack to rack traffic. Color intensity is log-scale and normalized.  
Ghobadi et al., SIGCOMM'16

# Enablers for Reconfigurable DCNs

## Programmable Physical Layer

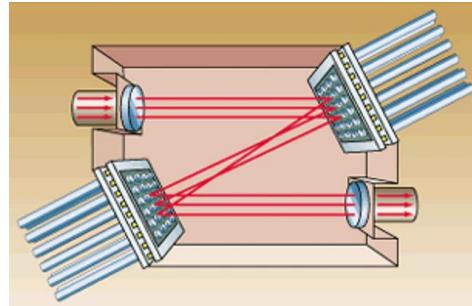


Better power consumption, fan-out, “rate free”, at the cost of reconfiguration times & point-point connectivity

Images taken from the respective papers

# Overview of Technological Enablers

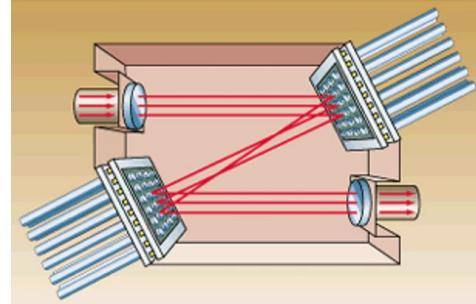
- Microelectromechanical systems (MEMS)
  - In 3D:  $2N$  mirrors can connect  $N$  input to  $N$  output ports
  - In 2D: Less connectivity, but faster



<https://www.laserfocusworld.com/optics/article/16556781/many-approaches-taken-for-alloptical-switching> (Hecht, 2001)

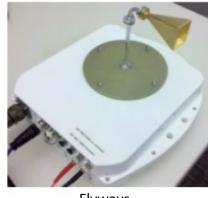
# Overview of Technological Enablers

- Microelectromechanical systems (MEMS)
  - In 3D:  $2N$  mirrors can connect  $N$  input to  $N$  output ports
  - In 2D: Less connectivity, but faster



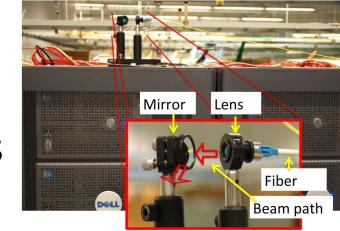
<https://www.laserfocusworld.com/optics/article/16556781/many-approaches-taken-for-alloptical-switching> (Hecht, 2001)

- (Beamformed) Wireless



Flyways

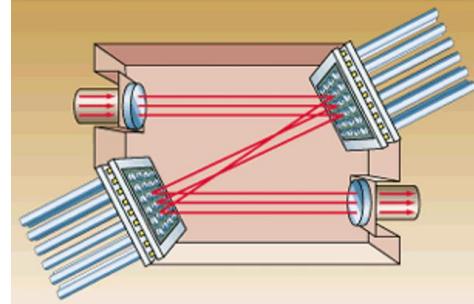
and Free-Space Optics



FireFly

# Overview of Technological Enablers

- Microelectromechanical systems (MEMS)
  - In 3D:  $2N$  mirrors can connect  $N$  input to  $N$  output ports
  - In 2D: Less connectivity, but faster

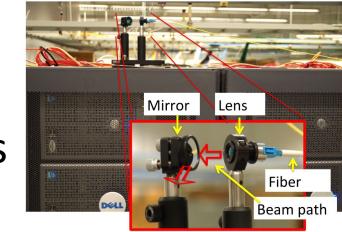


<https://www.laserfocusworld.com/optics/article/16556781/many-approaches-taken-for-alloptical-switching> (Hecht, 2001)

- (Beamformed) Wireless



and Free-Space Optics

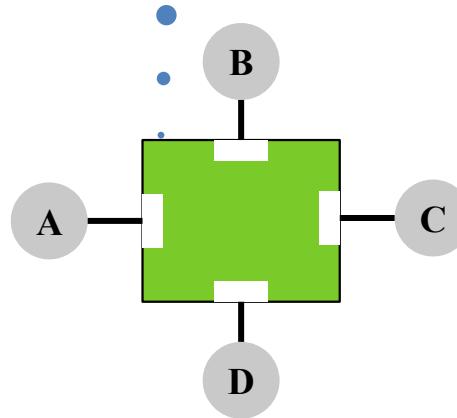


- Connection medium can also often be shared, e.g., different wavelengths

# It's a Match(ing)!

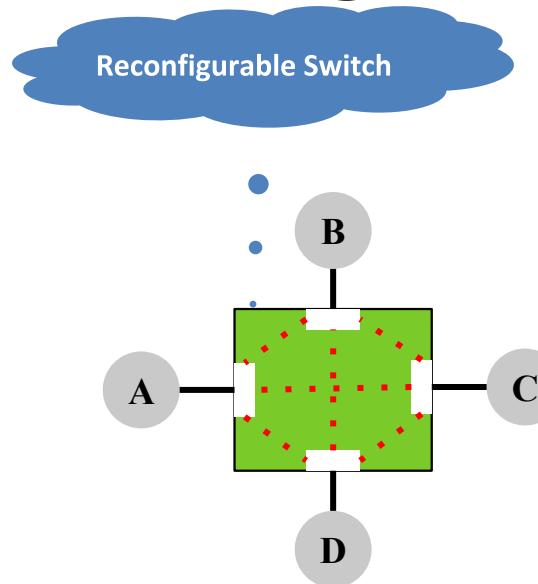
- Idea: Create “physical” connections

Reconfigurable Switch



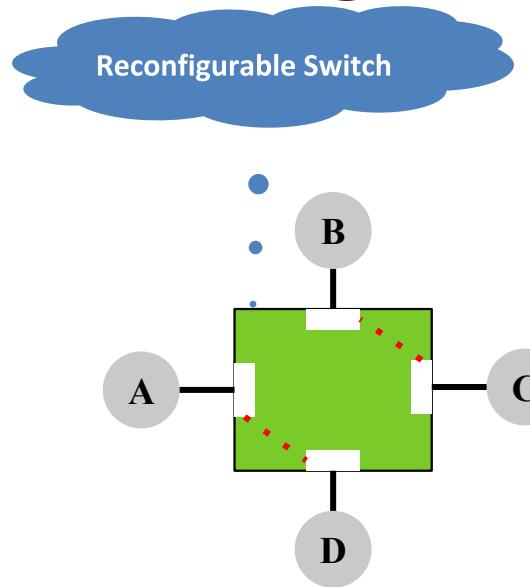
# It's a Match(ing)!

- Idea: Create “physical” connections
  - Difference: Not all-to-all switch
    - E.g. just 1 connection per node



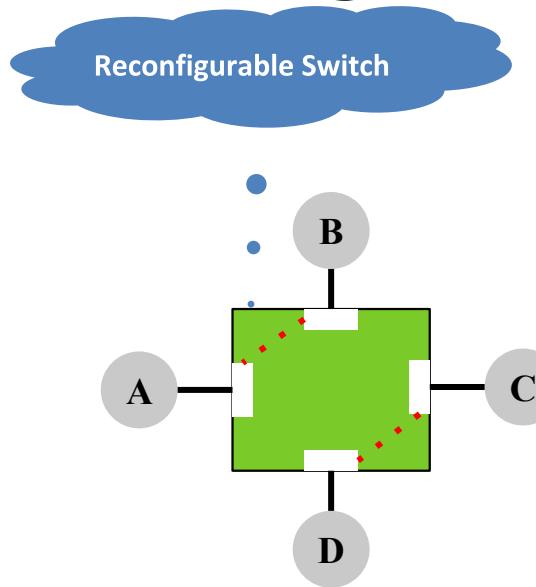
# It's a Match(ing)!

- Idea: Create “physical” connections
  - Difference: Not all-to-all switch
    - E.g. just 1 connection per node



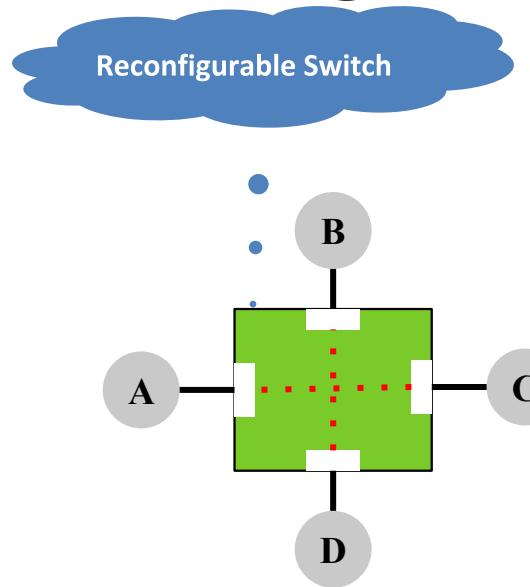
# It's a Match(ing)!

- Idea: Create “physical” connections
  - Difference: Not all-to-all switch
    - E.g. just 1 connection per node



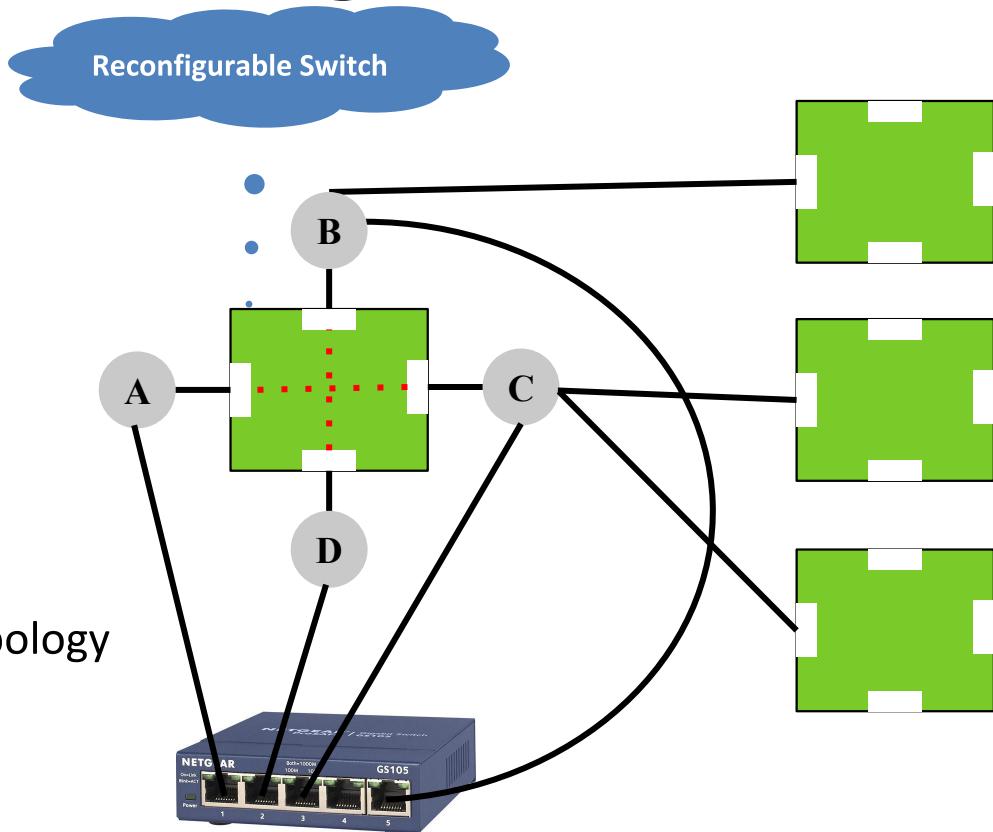
# It's a Match(ing)!

- Idea: Create “physical” connections
  - Difference: Not all-to-all switch
    - E.g. just 1 connection per node



# It's a Match(ing)!

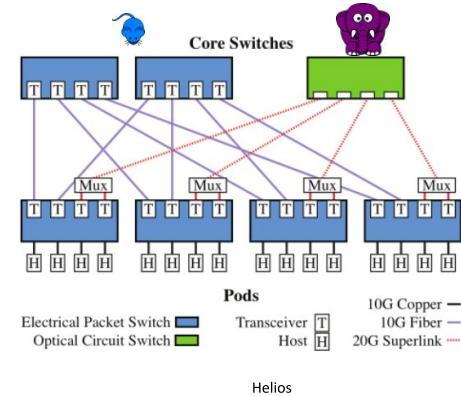
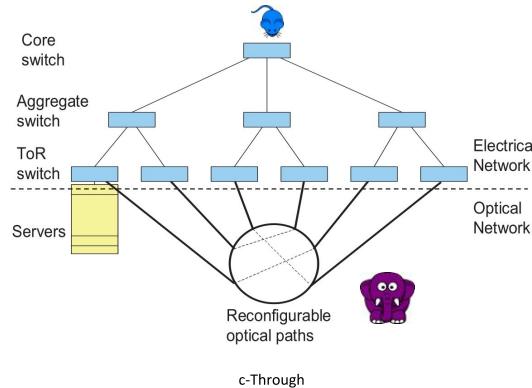
- Idea: Create “physical” connections
  - Difference: Not all-to-all switch
    - E.g. just 1 connection per node
    - Or many more than 1
    - Or separated sender/receiver
- Basic connectivity often by static topology
  - Hybrid: Static+Reconfigurable



# Helios/c-Through

(Farrington et al. / Wang et al. @SIGCOMM 2010)

- Layout:
  - DCN-Topology + optical circuit switch (OCS,  $\sim O(10)\text{ms}$ )



# Helios/c-Through

(Farrington et al. / Wang et al. @SIGCOMM 2010)

- Algorithmic Idea:
  - Estimate demand between racks/pods
  - Pose as matching problem (throughput/volume per epoch as weights)
    - Unidirectional in Helios
  - Compute maximum matching (eg with Edmond's)



# Helios/c-Through

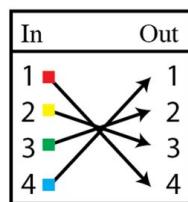
(Farrington et al. / Wang et al. @SIGCOMM 2010)

- Algorithmic Idea:
  - Estimate demand between racks/pods
  - Pose as matching problem (throughput/volume per epoch as weights)
    - Unidirectional in Helios
  - Compute maximum matching (eg with Edmond's)
- Example from Helios with 4 pods and link capacity of 4

Demand Matrix i

|   | 1                | 2 | 3 | 4                |
|---|------------------|---|---|------------------|
| 1 | 0                | 1 | 1 | 3                |
| 2 | 7 <sub>(4)</sub> | 0 | 3 | 1                |
| 3 | 1                | 3 | 0 | 9 <sub>(4)</sub> |
| 4 | 3                | 2 | 1 | 0                |

Circuit Switch i



# Helios/c-Through

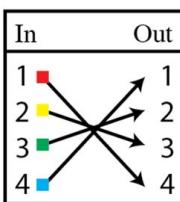
(Farrington et al. / Wang et al. @SIGCOMM 2010)

- Algorithmic Idea:
  - Estimate demand between racks/pods
  - Pose as matching problem (throughput/volume per epoch as weights)
    - Unidirectional in Helios
  - Compute maximum matching (eg with Edmond's)
- Example from Helios with 4 pods and link capacity of 4

Demand Matrix i

|   | 1         | 2 | 3 | 4         |
|---|-----------|---|---|-----------|
| 1 | 0         | 1 | 1 | 3         |
| 2 | $7_{(4)}$ | 0 | 3 | 1         |
| 3 | 1         | 3 | 0 | $9_{(4)}$ |
| 4 | 3         | 2 | 1 | 0         |

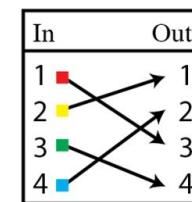
Circuit Switch i



Demand Matrix i+1

|   | 1         | 2 | 3 | 4         |
|---|-----------|---|---|-----------|
| 1 | 0         | 1 | 1 | 0         |
| 2 | $7_{(4)}$ | 0 | 0 | 1         |
| 3 | 1         | 0 | 0 | $9_{(4)}$ |
| 4 | 0         | 2 | 1 | 0         |

Circuit Switch i+1



# Helios/c-Through

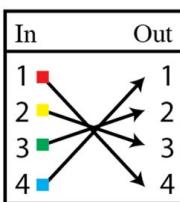
(Farrington et al. / Wang et al. @SIGCOMM 2010)

- Algorithmic Idea:
  - Estimate demand between racks/pods
  - Pose as matching problem (throughput/volume per epoch as weights)
    - Unidirectional in Helios
  - Compute maximum matching (eg with Edmond's)
- Example from Helios with 4 pods and link capacity of 4

Demand Matrix i

|   | 1         | 2 | 3 | 4         |
|---|-----------|---|---|-----------|
| 1 | 0         | 1 | 1 | 3         |
| 2 | $7_{(4)}$ | 0 | 3 | 1         |
| 3 | 1         | 3 | 0 | $9_{(4)}$ |
| 4 | 3         | 2 | 1 | 0         |

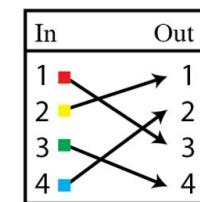
Circuit Switch i



Demand Matrix i+1

|   | 1         | 2 | 3 | 4         |
|---|-----------|---|---|-----------|
| 1 | 0         | 1 | 1 | 0         |
| 2 | $7_{(4)}$ | 0 | 0 | 1         |
| 3 | 1         | 0 | 0 | $9_{(4)}$ |
| 4 | 0         | 2 | 1 | 0         |

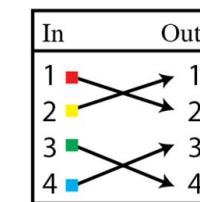
Circuit Switch i+1



Demand Matrix i+2

|   | 1         | 2 | 3 | 4         |
|---|-----------|---|---|-----------|
| 1 | 0         | 1 | 0 | 0         |
| 2 | $3_{(4)}$ | 0 | 0 | 1         |
| 3 | 1         | 0 | 0 | $5_{(4)}$ |
| 4 | 0         | 0 | 1 | 0         |

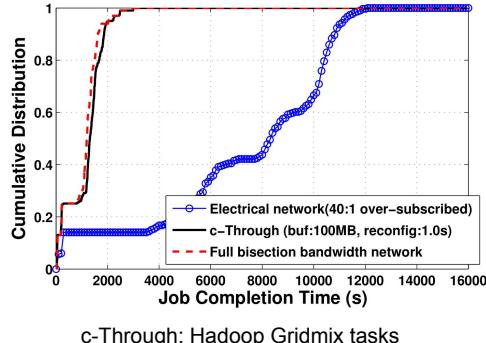
Circuit Switch i+2



# Helios/c-Through

(Farrington et al. / Wang et al. @SIGCOMM 2010)

- Algorithmic Idea:
  - Estimate demand between racks/pods
  - Pose as matching problem (throughput/volume per epoch as weights)
    - Unidirectional in Helios
  - Compute maximum matching (eg with Edmond's)
- Performance example for data-intensive tasks:



# Routing Policy Restrictions

- So far: Routing options restricted to single-hop



# Routing Policy Restrictions

- So far: Routing options restricted to single-hop



# Routing Policy Restrictions

- So far: Routing options restricted to single-hop



# Routing Policy Restrictions

- So far: Routing options restricted to single-hop



# Routing Policy Restrictions

- So far: Routing options restricted to single-hop



# Routing Policy Restrictions

- So far: Routing options restricted to single-hop



# Routing Policy Restrictions

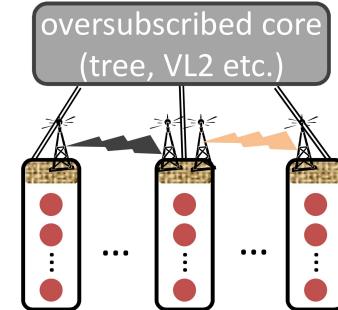
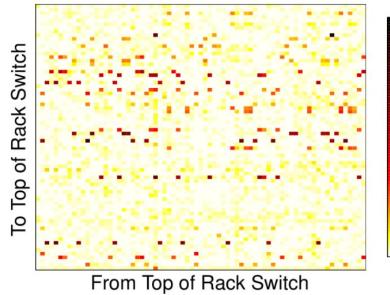
- So far: Routing options restricted to single-hop



# Flyways

(Kandula et al., HotNets 2009/SIGCOMM 2011)

- Idea: Tackle hotspots by adding so-called flyways
  - Directional wireless (60GHz) [or also 802.11n/new static links]
    - $\sim O(10)$ ms (also propose the use of phased arrays, delay in microseconds)



# Flyways

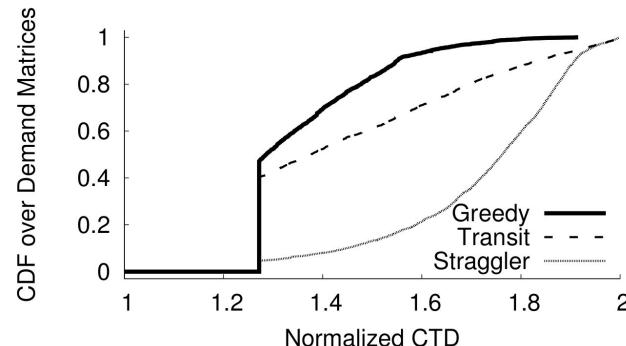
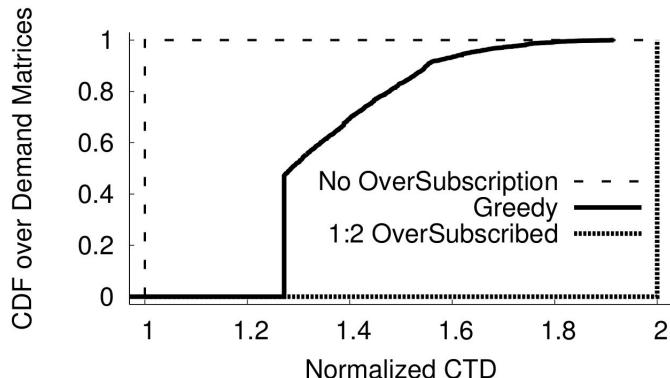
(Kandula et al., HotNets 2009/SIGCOMM 2011)

- Idea: Tackle hotspots by adding so-called flyways
  - Directional wireless (60GHz) [or also 802.11n/new static links]
- How to choose new links?
  - Three preliminary different strategies proposed w.r.t. completion time:
    - Optimization program
      - downside: intractable
    - Stragglers: help biggest demands on bottleneck links
      - downside: might not help much per bottleneck, fan-out/in of demands
    - Allow transit on straggler links
      - helps, but not optimized for it

# Flyways

(Kandula et al., HotNets 2009/SIGCOMM 2011)

- Final strategy: **Greedy** with transit
  - Consider bottleneck link connected to ToR
    - Take flyway link that helps most w.r.t.
      - deviating traffic from bottleneck link + allowing transit



Beats other tractable strategies with one device per ToR

Images taken from the respective papers

# Mirror Mirror on the Ceiling

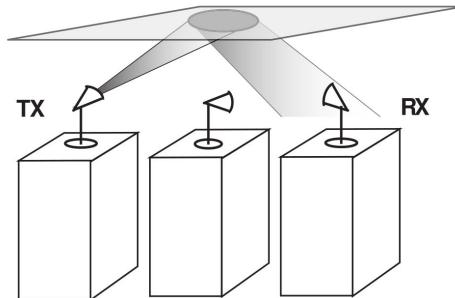
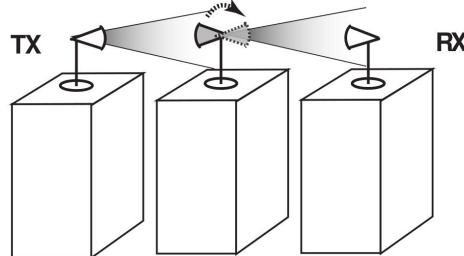
(Zhou et al., SIGCOMM 2012)

- Problem in Flyways: Limited any-to-any connections
  - Transmission might be blocked by obstacles (already 2.5mm is bad)
  - Radio interference between links
  - In combination limits many possible flyway configurations

# Mirror Mirror on the Ceiling

(Zhou et al., SIGCOMM 2012)

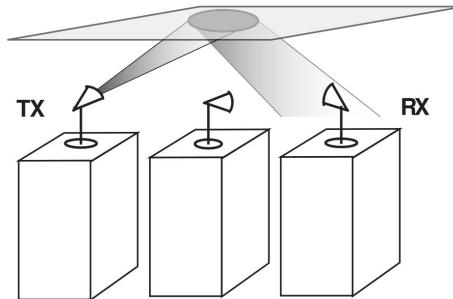
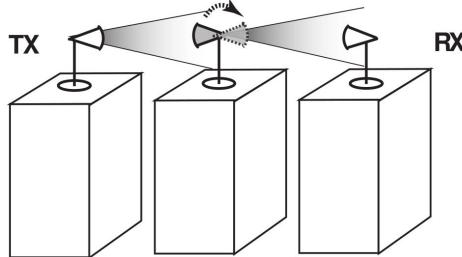
- Problem in Flyways: Limited any-to-any connections
  - Transmission might be blocked by obstacles (already 2.5mm is bad)
  - Radio interference between links
  - In combination limits many possible flyway configurations
- Approach from Zhou et al.: Go from 2D to 3D



# Mirror Mirror on the Ceiling

(Zhou et al., SIGCOMM 2012)

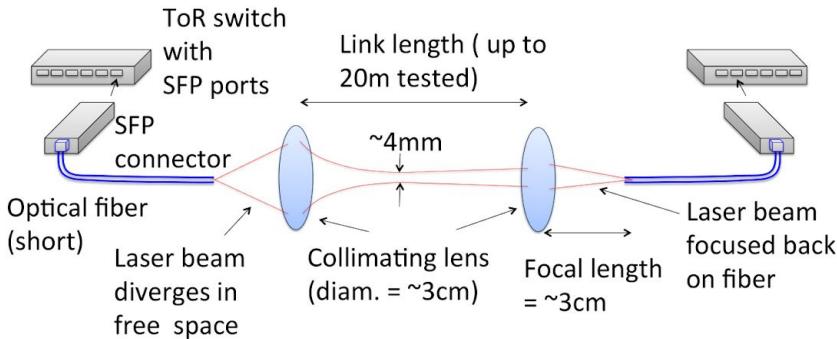
- Algorithmic/technical idea:
  - Leverage multiple radios per rack and multiple 60GHz frequency channels
  - Keep Signal-to-Interference-Noise-Ratio (SINR) in mind for conflicts
    - Greedily schedule requests ordered by conflict degrees
      - No preemption, no multi-hop



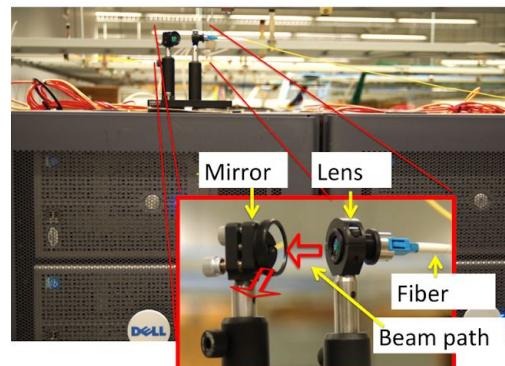
# FireFly

(Hamedazimi et al., SIGCOMM 2012)

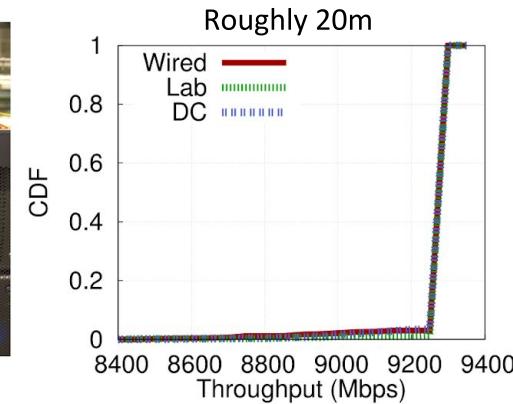
- Even beamformed wireless still suffers from interference/range
- Idea: Use Free-Space Optics (FSO)
  - O(10)ms steering, O(10) fan-out



(a) FSO link design



(b) DC set up

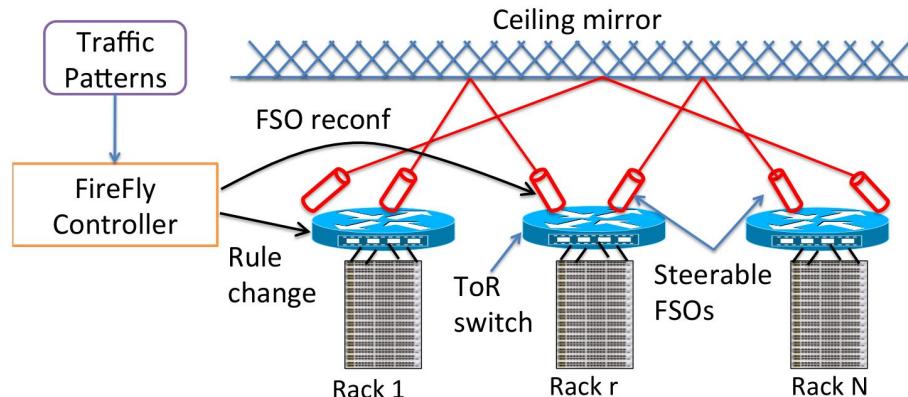


(c) TCP throughput

# FireFly

(Hamedazimi et al., SIGCOMM 2012)

- Bring concept from *Mirror, Mirror* to FSO
- Core algorithmic idea:
  - Periodically recompute topology for throughput (optim. formulation)
  - Greedily augment current matching to shorten routes for eg new 
  - (don't disconnect)



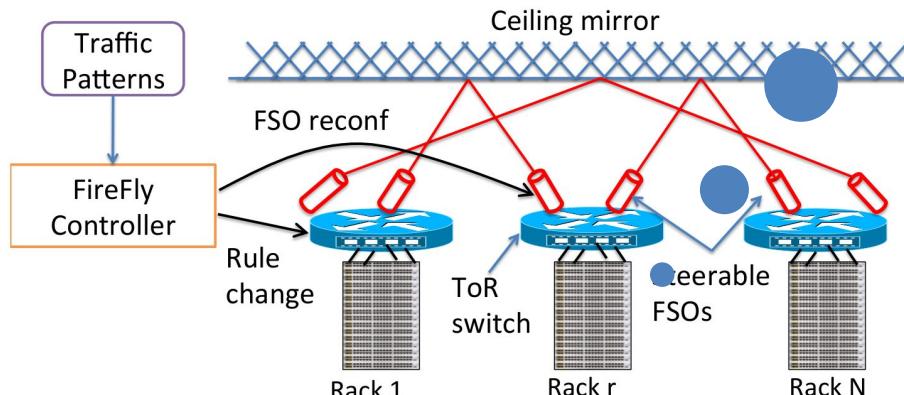
# FireFly

(Hamedazimi et al., SIGCOMM 2012)

- Bring concept from *Mirror, Mirror* to life
- Core algorithmic idea:
  - Periodically recompute topology for throughput
  - Greedily augment current matching to stay connected
    - (don't disconnect)

More on FSO later in ProjecToR

- Distributed algorithm
- Microsecond reconfigurations
  - Programmable mirrors
- $O(1000)$  fan-out



# Proteus/OSA

(Singla et al., HotNets 2010/NSDI 2012/ ToN 2014)

- More thoughts on all-reconfigurable:
  - What if traffic is extremely dynamic?
    - Prior studies show: Usually somewhat stable for many DCN applications
  - Recall:
    - Need multi-hop connections

# Proteus/OSA

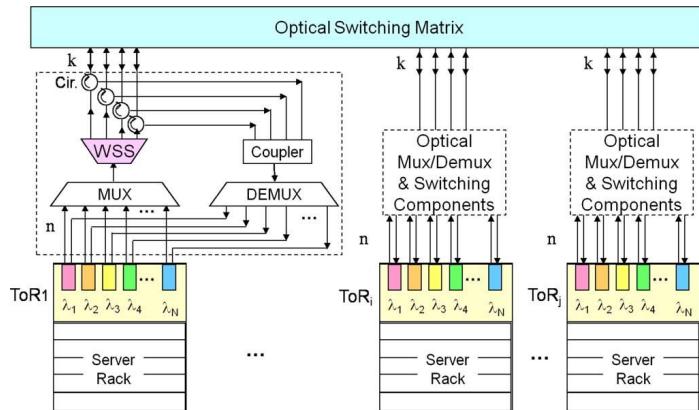
(Singla et al., HotNets 2010/NSDI 2012/ ToN 2014)

- More thoughts on all-reconfigurable:
  - What if traffic is extremely dynamic?
    - Prior studies show: Usually somewhat stable for many DCN applications
  - Recall:
    - Need multi-hop connections
- Multi-hop and single connection to OCS?
  - Not so good in all-reconfigurable setting
    - No connectivity (even with 2: requires Hamiltonian Cycle)
    - No scaling of link capacities

# Proteus/OSA

(Singla et al., HotNets 2010/NSDI 2012/ ToN 2014)

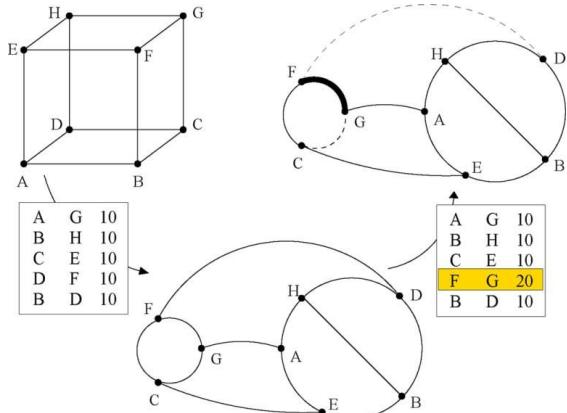
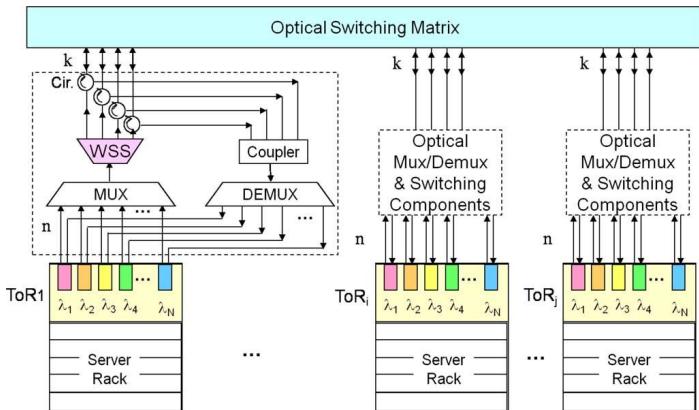
- Creating connectivity:
  - Multiple connections per rack



# Proteus/OSA

(Singla et al., HotNets 2010/NSDI 2012/ ToN 2014)

- Creating connectivity:
  - Multiple connections per rack
- Scaling link capacities
  - Allow “parallel” links



# Proteus/OSA

(Singla et al., HotNets 2010/NSDI 2012/ ToN 2014)

- Algorithmic idea:
  - Optimize for  , make  feasible

# Proteus/OSA

(Singla et al., HotNets 2010/NSDI 2012/ ToN 2014)

- Algorithmic idea:
    - Optimize for  , make  feasible
1.  For  $b$  connections per rack, leverage  $b$ -matching algorithms
- a. Estimated demands as weights (Poly-time solvable, efficient heuristics exist)

# Proteus/OSA

(Singla et al., HotNets 2010/NSDI 2012/ ToN 2014)

- Algorithmic idea:
    - Optimize for  , make  feasible
1.  For  $b$  connections per rack, leverage  $b$ -matching algorithms
    - a. Estimated demands as weights (Poly-time solvable, efficient heuristics exist)
  2.  Use link-exchanges to make topology connected
    - a. Heuristic: Connect 2 components by removing low weight links

# Proteus/OSA

(Singla et al., HotNets 2010/NSDI 2012/ ToN 2014)

- Algorithmic idea:
    - Optimize for  , make  feasible
1.  For  $b$  connections per rack, leverage  $b$ -matching algorithms
    - a. Estimated demands as weights (Poly-time solvable, efficient heuristics exist)
  2.  Use link-exchanges to make topology connected
    - a. Heuristic: Connect 2 components by removing low weight links
  3. Deploy routing (eg shortest paths)

# Proteus/OSA

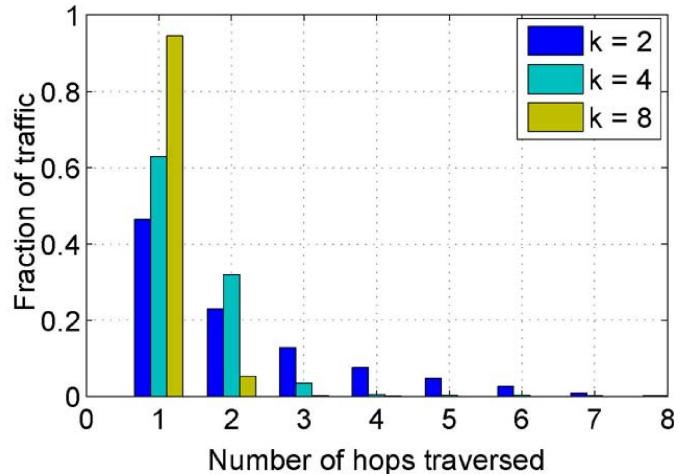
(Singla et al., HotNets 2010/NSDI 2012/ ToN 2014)

- Algorithmic idea:
  - Optimize for  , make  feasible
- 1.  For  $b$  connections per rack, leverage  $b$ -matching algorithms
  - a. Estimated demands as weights (Poly-time solvable, efficient heuristics exist)
- 2.  Use link-exchanges to make topology connected
  - a. Heuristic: Connect 2 components by removing low weight links
- 3. Deploy routing (eg shortest paths)
- 4.  Assign wavelengths along links to distribute capacity
  - a. Each link at least one wavelength color, no color connected twice to a node
  - b. Solved by link-coloring on multigraphs (efficient heuristics exist)

# Proteus/OSA

(Singla et al., HotNets 2010/NSDI 2012/ ToN 2014)

- Impact of degree on hop-counts (OSA uses  $k=4$  in most evaluations)
- Using Mapreduce-demands with 80 ToRs



# Average Path Length

(Foerster et al., ANCS 2018, SIGCOMM CCR 2019, Networking 2019)

- So far: Optimizing for throughput
  - Recall last slide: short paths are desired
    - Also used in FireFly for new elephants
  - Respectively: “**bandwidth tax**” (Mellete et al., 2019)

# Average Path Length

(Foerster et al., ANCS 2018, SIGCOMM CCR 2019, Networking 2019)

- So far: Optimizing for throughput
  - Recall last slide: short paths are desired
    - Also used in FireFly for new elephants
  - Respectively: “**bandwidth tax**” (Mellete et al., 2019)
- Different objective:
  - **Minimize (weighted) average path length**
  - Popular in many fields, e.g., OSPF

# Average Path Length

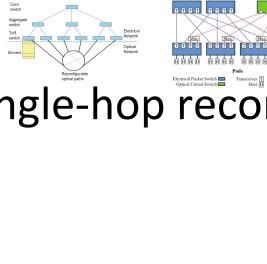
(Foerster et al., ANCS 2018, SIGCOMM CCR 2019, Networking 2019)

- So far: Optimizing for throughput
  - Recall last slide: short paths are desired
    - Also used in FireFly for new elephants
  - Respectively: “**bandwidth tax**” (Mellete et al., 2019)
- Different objective:
  - **Minimize (weighted) average path length**
  - Popular in many fields, e.g., OSPF
- How difficult from an algorithmic perspective?

# Average Path Length

(Foerster et al., ANCS 2018, SIGCOMM CCR 2019, Networking 2019)

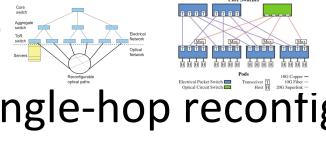
- In model from Helios/c-Through?
  - Recall: packet switched network XOR single-hop reconfigurable (eg OCS)



# Average Path Length

(Foerster et al., ANCS 2018, SIGCOMM CCR 2019, Networking 2019)

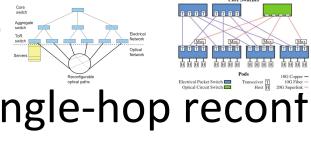
- In model from Helios/c-Through?
  - Recall: packet switched network XOR single-hop reconfigurable (eg OCS)
- Good algorithmic news:
  - Efficiently polynomial-time solvable (weighted matching algorithms)
  - Also for many connections per node to the OCS



# Average Path Length

(Foerster et al., ANCS 2018, SIGCOMM CCR 2019, Networking 2019)

- In model from Helios/c-Through?
  - Recall: packet switched network XOR single-hop reconfigurable (eg OCS)
- Good algorithmic news:
  - Efficiently polynomial-time solvable (weighted matching algorithms)
  - Also for many connections per node to the OCS
- Bad algorithmic news:
  - Such a restriction is self-imposed and hurts performance



# Average Path Length

(Foerster et al., ANCS 2018, SIGCOMM CCR 2019, Networking 2019)

- How hard to be optimal after lifting restrictions?
  - Already “simple” settings are NP-hard
    - E.g., each route may use at *most one reconfigurable link*



# Average Path Length

(Foerster et al., ANCS 2018, SIGCOMM CCR 2019, Networking 2019)

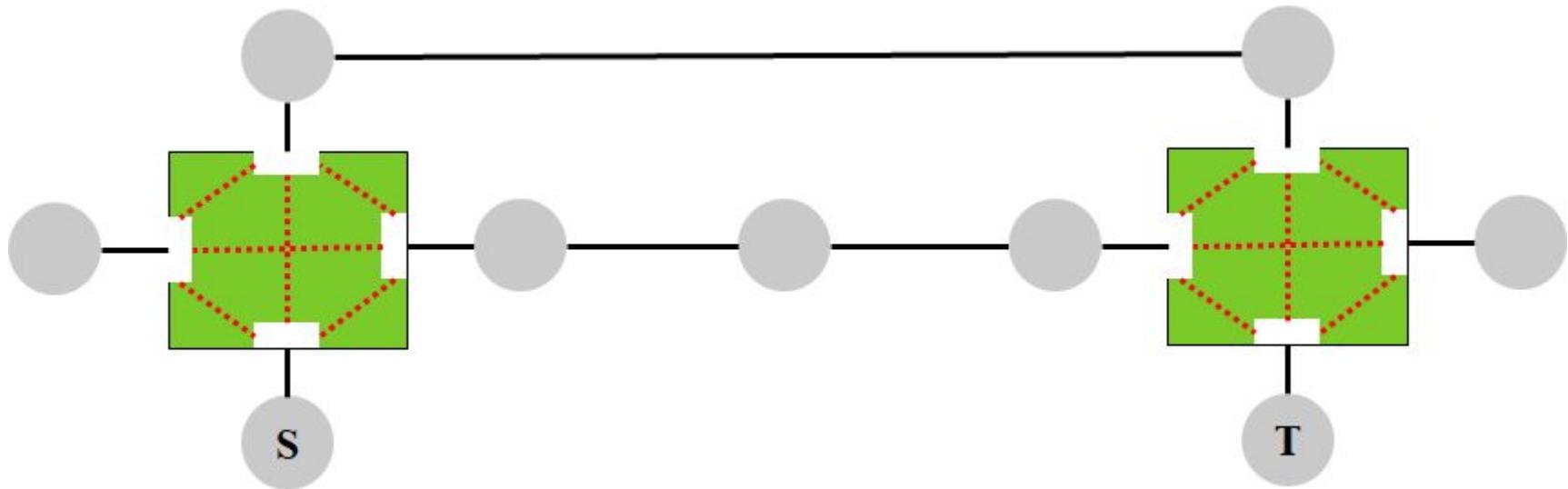
- How hard to be optimal after lifting restrictions?
  - Already “simple” settings are NP-hard
    - E.g., each route may use at *most one reconfigurable link*
- But: We can lift Dijkstra’s algorithm into this setting\*
  - Each reconfigurable switch traversed at most once per flow



\*Assuming that for each reconfigurable switch holds: its link weights uphold the triangle equality, e.g., being identical

# Average Path Length

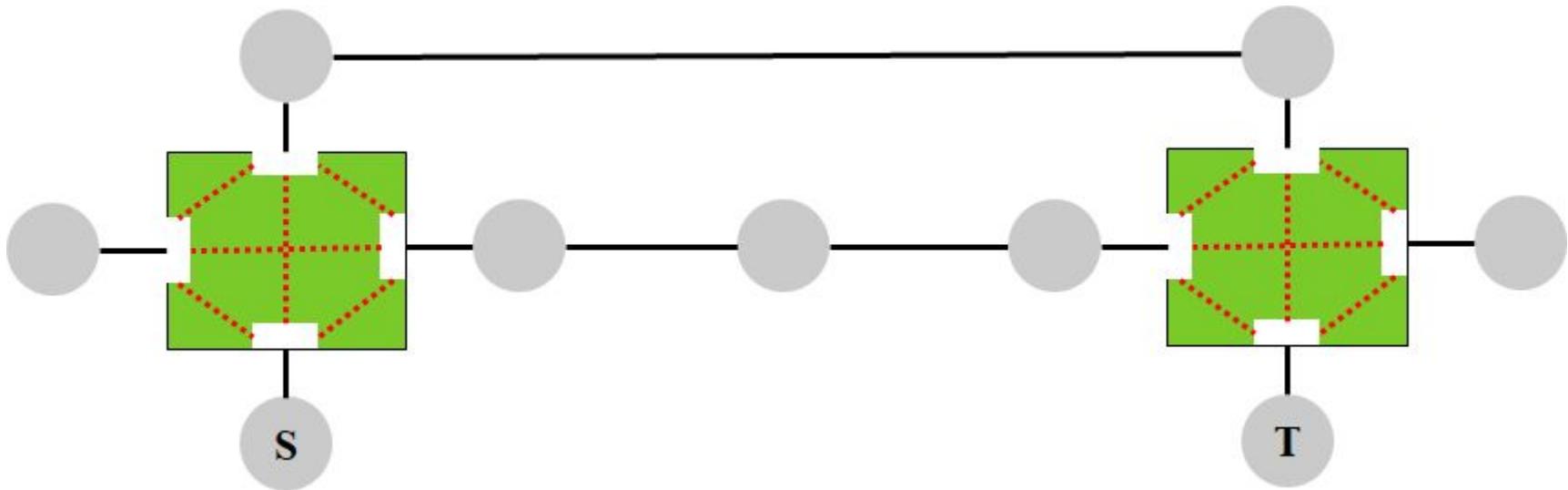
(Foerster et al., ANCS 2018, SIGCOMM CCR 2019, Networking 2019)



# Average Path Length

(Foerster et al., ANCS 2018, SIGCOMM CCR 2019, Networking 2019)

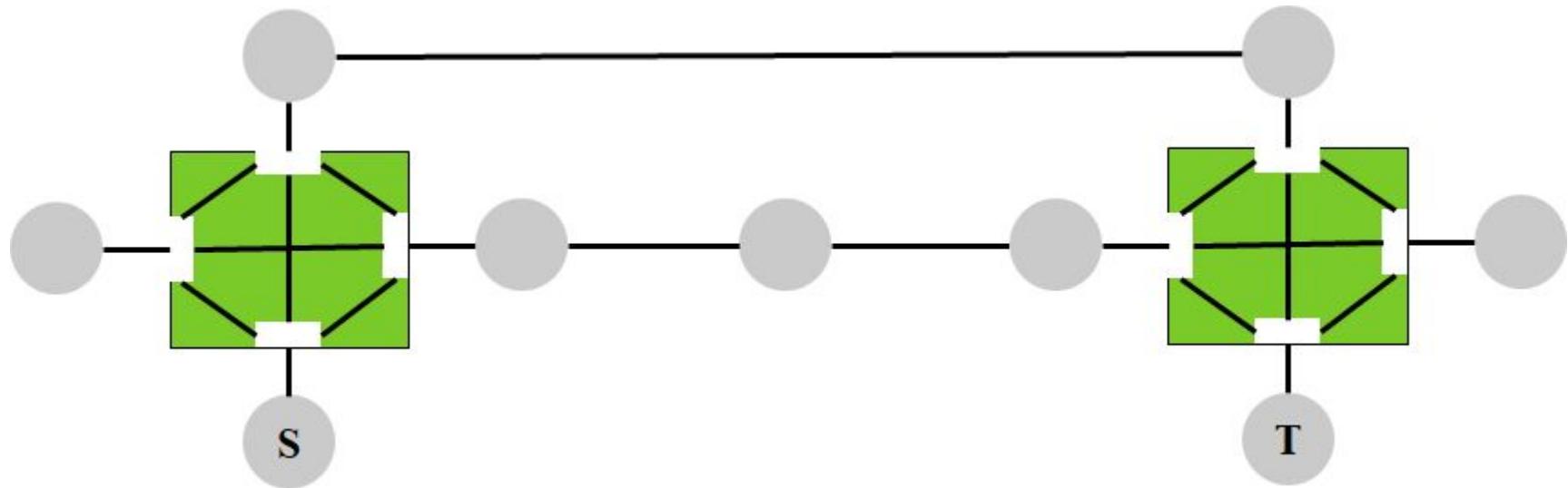
1. Add all still possible reconfigurable links as static links



# Average Path Length

(Foerster et al., ANCS 2018, SIGCOMM CCR 2019, Networking 2019)

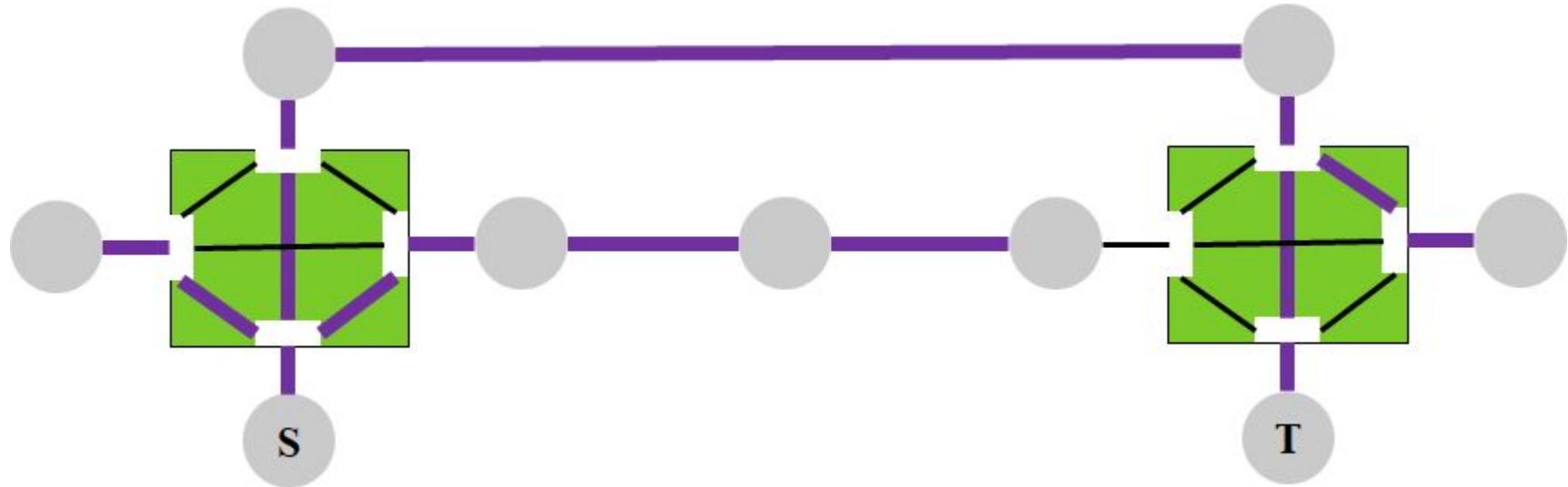
1. Add all still possible reconfigurable links as static links



# Average Path Length

(Foerster et al., ANCS 2018, SIGCOMM CCR 2019, Networking 2019)

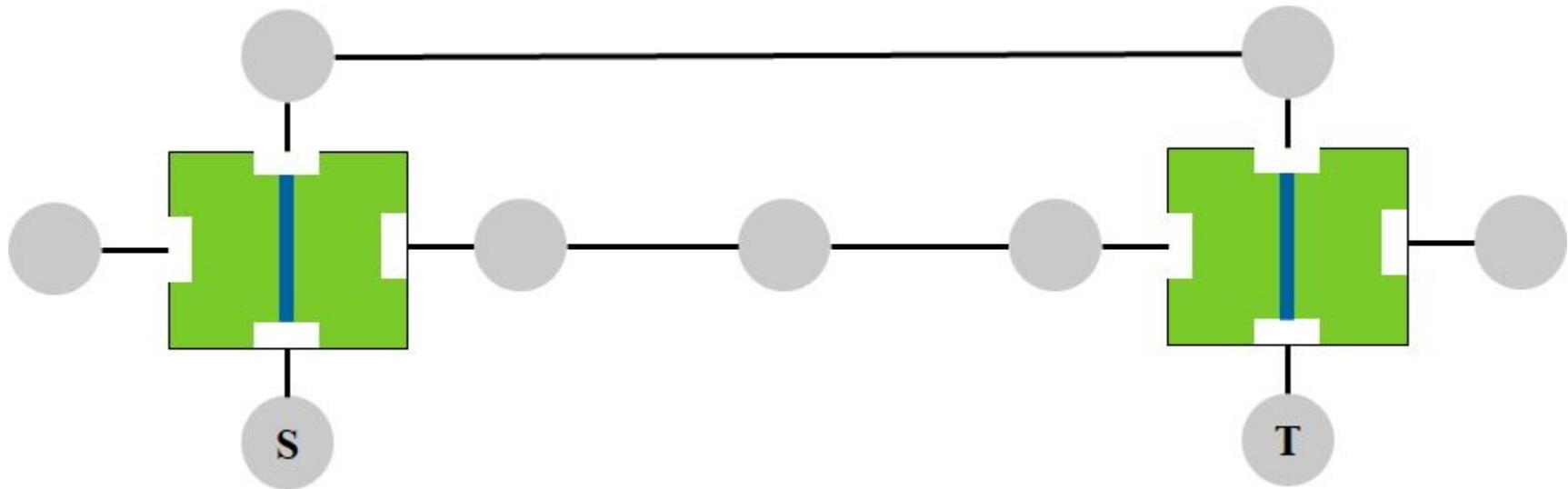
1. Add all still possible reconfigurable links as static links
2. **Run standard Dijkstra from source  $S$**



# Average Path Length

(Foerster et al., ANCS 2018, SIGCOMM CCR 2019, Networking 2019)

1. Add all still possible reconfigurable links as static links
2. Run standard Dijkstra from source  $S$
3. **Add newly used links on shortest path to  $T$  to the matchings**

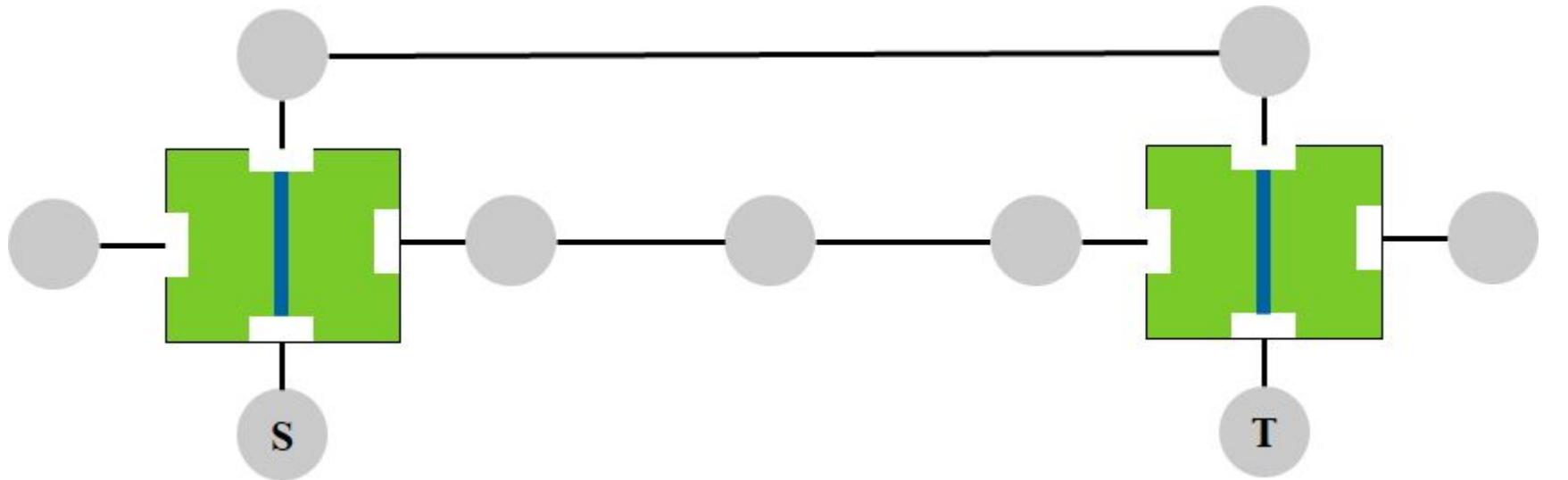


# Average Path Length

(Foerster et al., ANCS 2018, SIGCOMM CCR 2019, Networking 2019)

1. Add all still possible reconfigurable links as static links
2. Run standard Dijkstra from source  $S$
3. Add newly used links on shortest path to  $T$  to the matchings

Also works if some matching links already exist

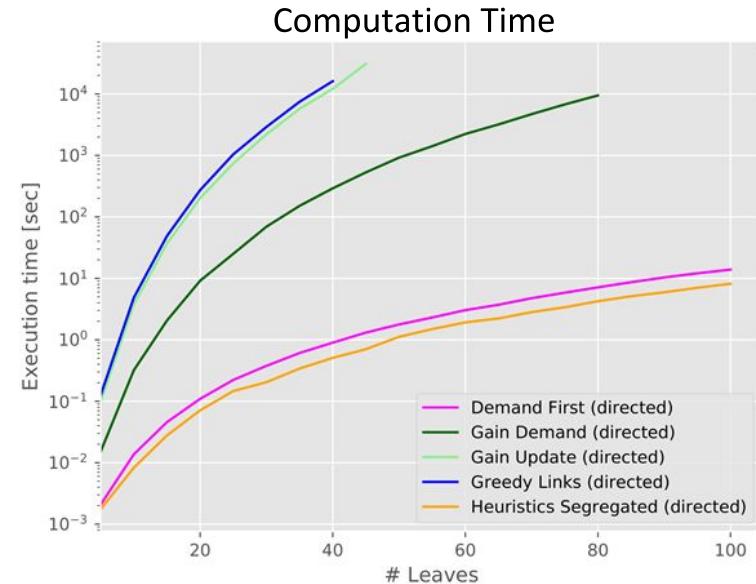
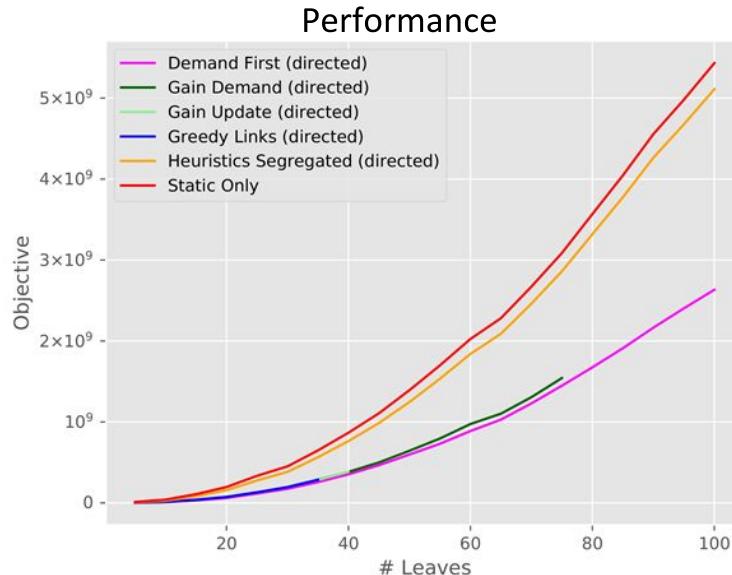


# Average Path Length

(Foerster et al., ANCS 2018, SIGCOMM CCR 2019, Networking 2019)

- Leverage for greedy heuristics (eg greedily run Dijkstra: )

 single hop matching baseline (optimal w.r.t. restricted segregated routing)



Traffic from recent  dataset, in fat-tree + OCS setting

# Mordia

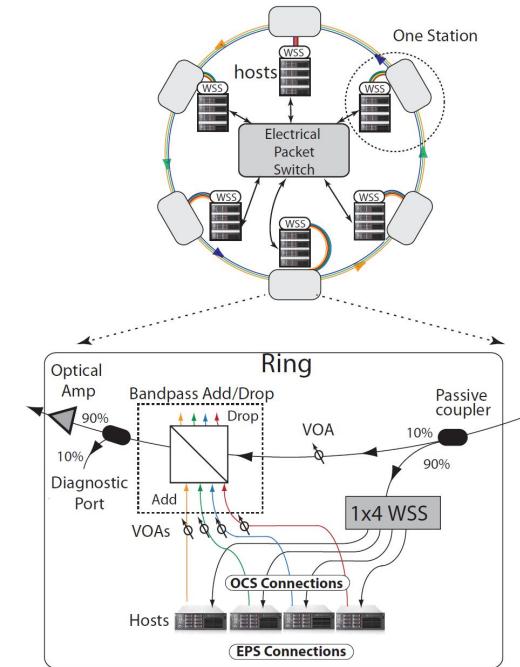
(Porter et al., SIGCOMM 2014)

- Back to the throughput objective

# Mordia

(Porter et al., SIGCOMM 2014)

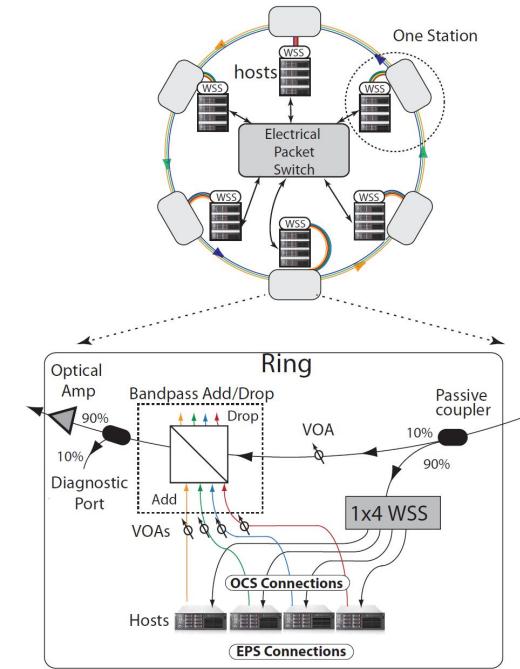
- Back to the throughput objective
- What if reconfiguration time goes
  - from milliseconds
  - to microseconds?



# Mordia

(Porter et al., SIGCOMM 2014)

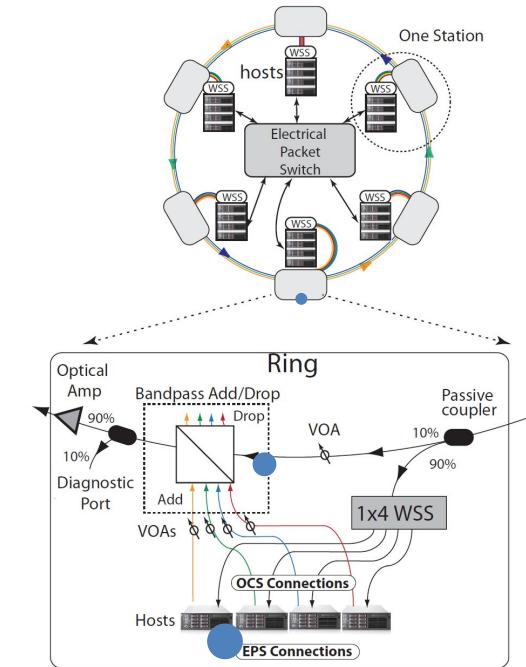
- Back to the throughput objective
- What if reconfiguration time goes
  - from milliseconds
  - to microseconds?
- Can't really compute well at microsecond scale?



# Mordia

(Porter et al., SIGCOMM 2014)

- Back to the throughput objective
- What if reconfiguration time goes
  - from milliseconds
  - to microseconds?
- Can't really compute well at microsecond scale?



Extended to multi-ring in  
Megaswitch (Chen et al., NSDI 2017)

# Mordia

(Porter et al., SIGCOMM 2014)

- Key idea:
  - Instead of a single reconfiguration...
  - ... compute a traffic matrix schedule (TMS)!

## Schedule

Detailed schedules are to be updated.

Go to schedule for: [Monday](#), [Tuesday](#), [Wednesday](#), [Thursday](#), [Friday](#)

Download the conference program in PDF format (TBA).

Monday, June 24th, 2019 (Tutorials)

| Time  | Tutorials  |
|-------|--|
| 08:15 | Continental Breakfast  |
| 09:00 | <b>Reconfigurable Networks: Enablers, Algorithms, Complexity (ReNets)</b>        |
| 09:30 | Ramakrishnan Durairajan, Klaus-T. Foerster, and Stefan Schmid<br>(Room 104A)     |
| 10:00 | The Power of SOAP Scheduling<br>Mor Harchol-Balter and Ziv Scully<br>(Room 104B) |
| 10:30 |  |
| 11:00 | Break  |
| 11:20 | FCRC Keynote:  |
| 12:00 | James E. Smith <a href="#">Abstract</a>  |
| 12:30 |  |
| 13:00 | Lunch  |
| 13:30 |  |
| 14:00 | <b>Two-Sided Marketplaces: An Algorithmic Viewpoint</b>                          |
| 14:30 | Sid Bannerjee and Yang Cai   |

# Mordia

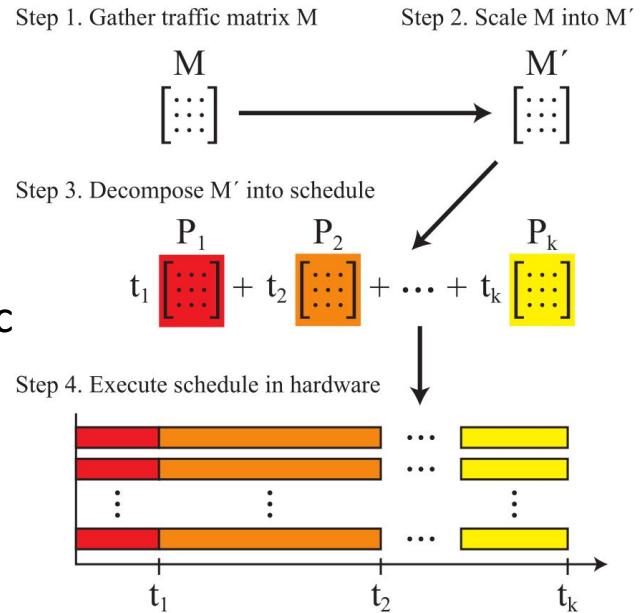
(Porter et al., SIGCOMM 2014)

- Traffic Matrix Scheduling
  - Look at all possible matchings
  - How much time to spend in each?

# Mordia

(Porter et al., SIGCOMM 2014)

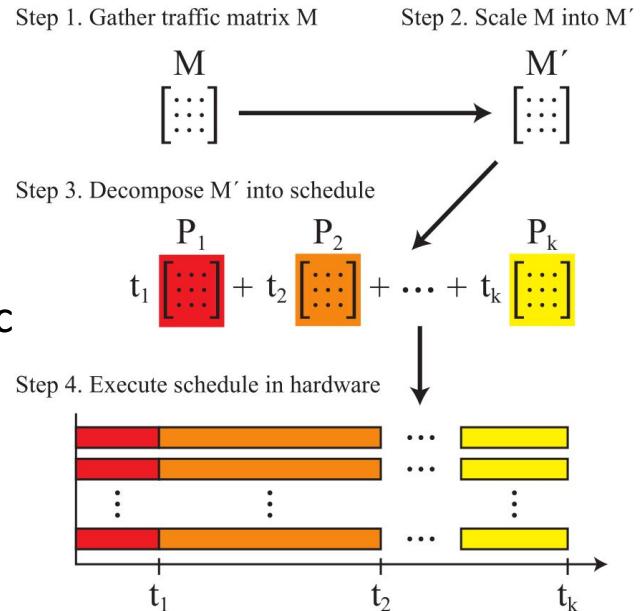
- Traffic Matrix Scheduling
  - Look at all possible matchings
  - How much time to spend in each?
- Idea: Single-hop, ideally fully utilize all links
  - Scale matrix s.t. is admissible & doubly-stochastic
    - Use Sinkhorn's algorithm, then
    - decompose with Birkhoff von Neumann decomposition ( $\sim O(n^2)$  runtime)



# Mordia

(Porter et al., SIGCOMM 2014)

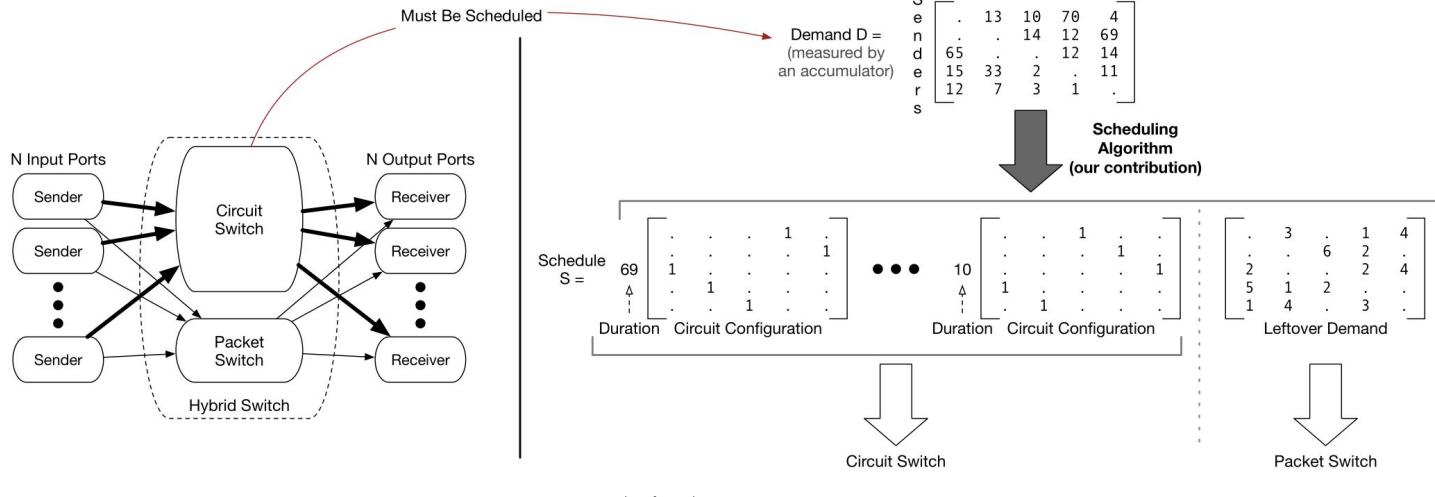
- Traffic Matrix Scheduling
  - Look at all possible matchings
  - How much time to spend in each?
- Idea: Single-hop, ideally fully utilize all links
  - Scale matrix s.t. is admissible & doubly-stochastic
    - Use Sinkhorn's algorithm, then
    - decompose with Birkhoff von Neumann decomposition ( $\sim O(n^2)$  runtime)
- Problem: Some slots extremely brief, up to  $O(n^2)$  many
  - Approximate by longest first, recompute after cut-off, tail into static network



# Solstice

(Liu et al., CoNEXT 2015)

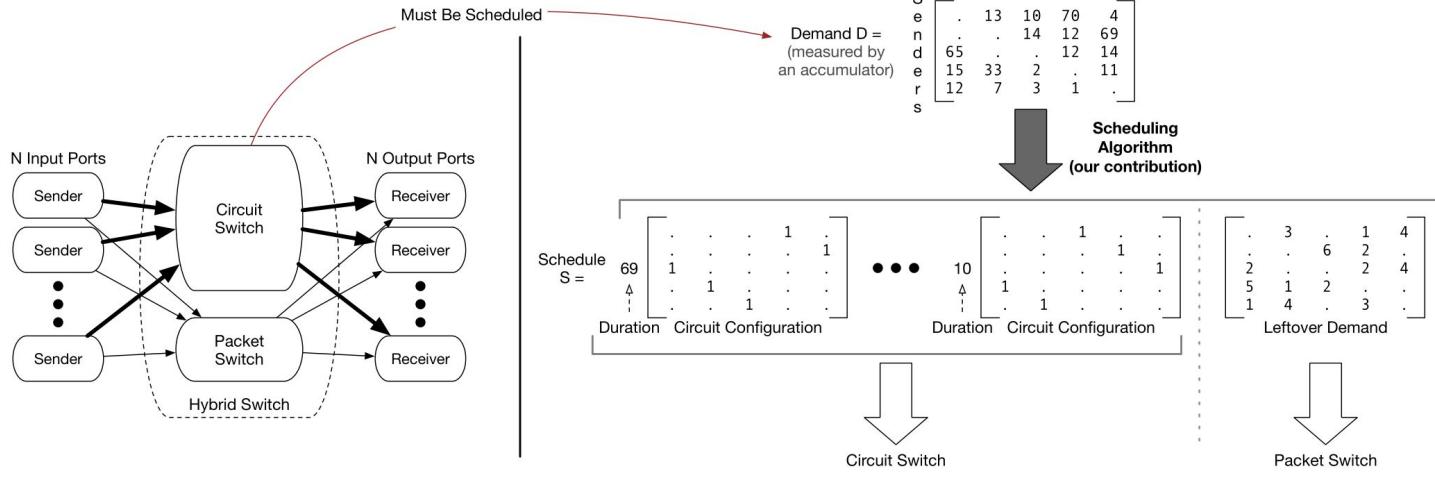
- So far: TMS does not take static network into account
  - Also not the reconfiguration time



# Solstice

(Liu et al., CoNEXT 2015)

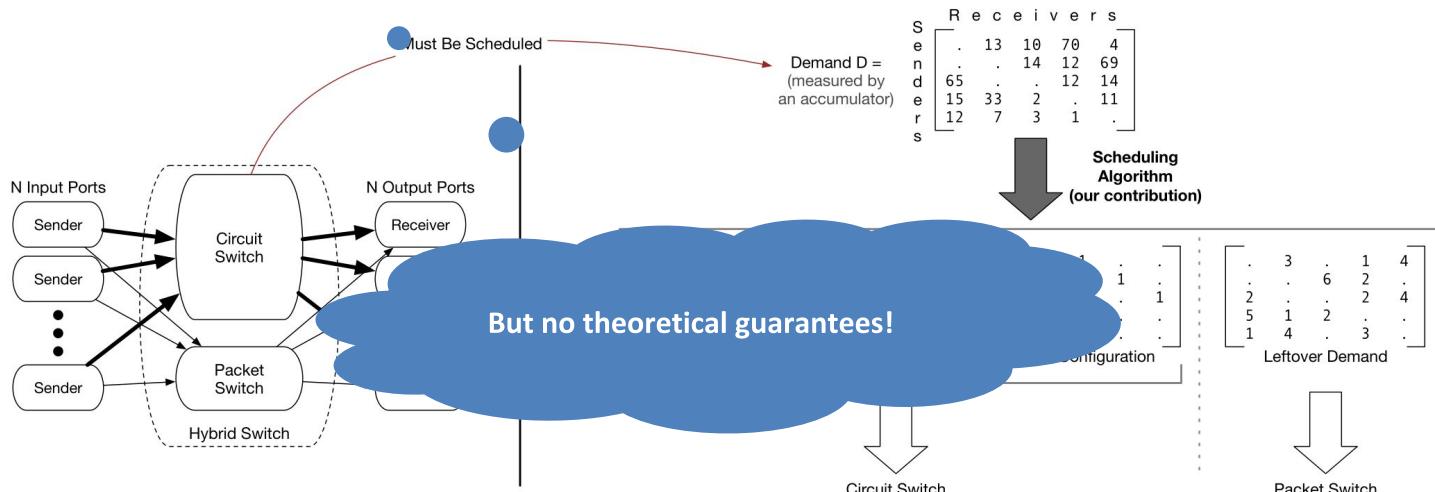
- Rough heuristic idea: Greedily schedule perfect matchings
  - Maximize minimum element in matching, repeat (outperforms BvN up to 2.9x)
    - About 14% away from optimal utilization, but runtime in  $\sim O(n^3)$



# Solstice

(Liu et al., CoNEXT 2015)

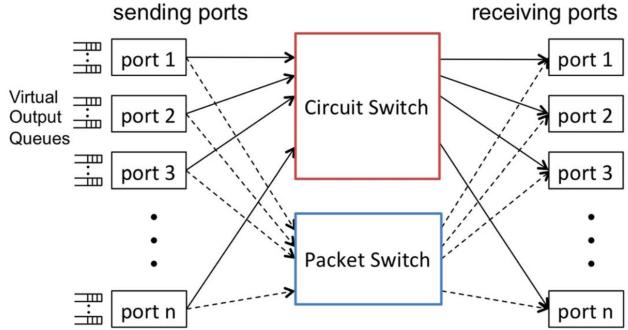
- Rough heuristic idea: Greedily schedule perfect matchings
  - Maximize minimum element in matching, repeat (outperforms BvN up to 2.9x)
    - About 14% away from optimal utilization, but runtime in  $\sim O(n^3)$



# Eclipse

(Venkatakrishnan et al., SIGMETRICS 2016 / Queuing Syst. 2018)

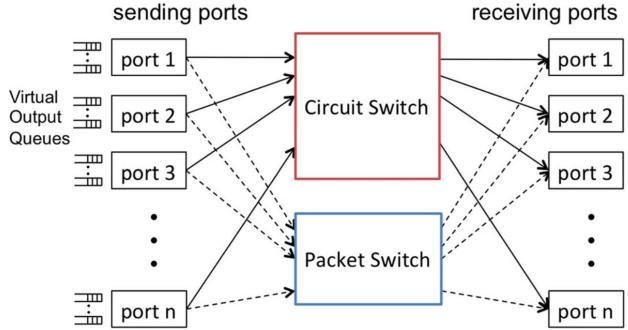
- Provides theoretical guarantees for TMS
  - hybrid switch model, reconfig. delay  $\delta$



# Eclipse

(Venkatakrishnan et al., SIGMETRICS 2016 / Queuing Syst. 2018)

- Provides theoretical guarantees for TMS
  - hybrid switch model, reconfig. delay  $\delta$
- Problem has submodular structure
  - Allows for  $(1-1/e)=\sim 0.63$  approximation via thresholding/max. weight matchings

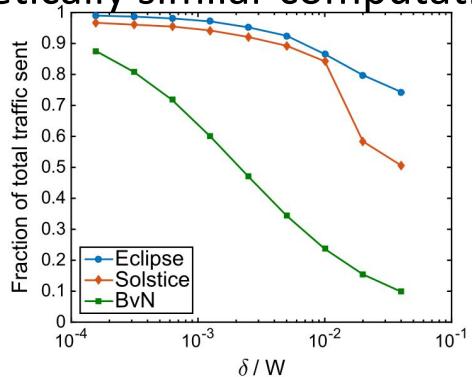


# Eclipse

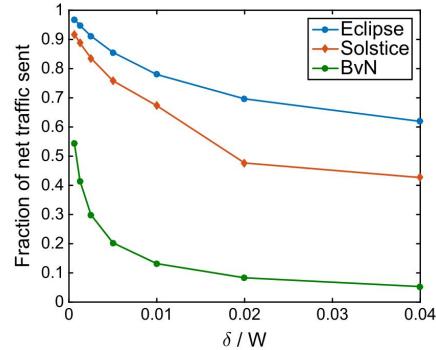
(Venkatakrishnan et al., SIGMETRICS 2016 / Queuing Syst. 2018)

- Provides theoretical guarantees for TMS
  - hybrid switch model, reconfig. delay  $\delta$
- Problem has submodular structure
  - Allows for  $(1-1/e) \approx 0.63$  approximation via thresholding/max. weight matchings
- Theoretically similar computation time to Solstice, performance for window W:

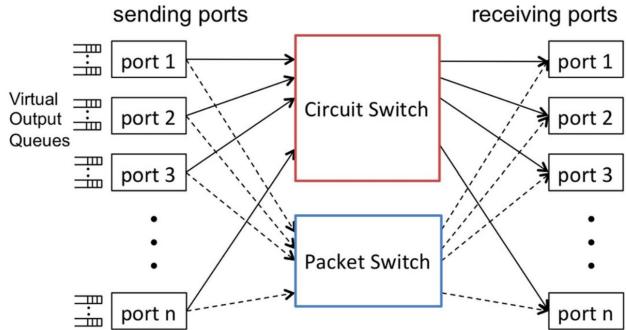
Sparse  
Skewed  
100 ports



Different sparsity and skew  
in submatrices  
200 ports



Images taken from the respective papers



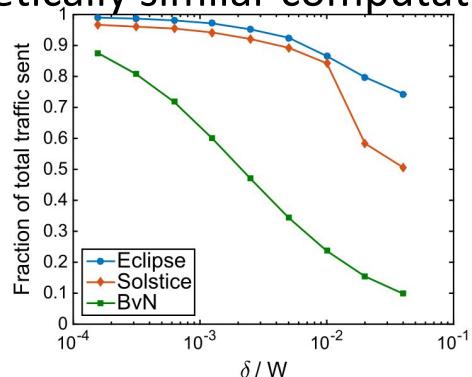
But no theoretical guarantees for multi-hop routing!

# Eclipse

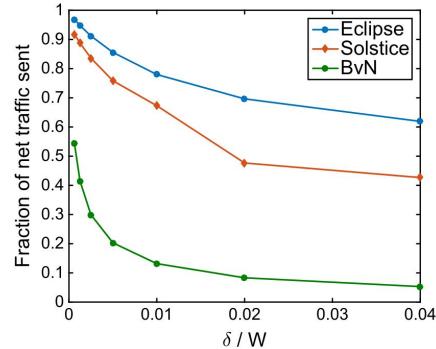
Wang et al., SIGMETRICS 2016 / Queueing Syst. 2018)

- Provides theoretical guarantees for TMS
  - hybrid switch model, reconfig. delay  $\delta$
- Problem has submodular structure.
  - Allows for  $(1-1/e) \approx 0.63$  approximation via thresholding/max. weight matchings
- Theoretically similar computation time to Solstice, performance for window  $W$ :

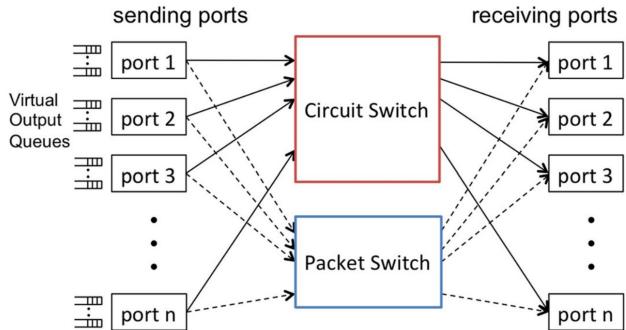
Sparse  
Skewed  
100 ports



Different sparsity and skew  
in submatrices  
200 ports



Images taken from the respective papers



# Computation Times

- Trade-off: Computation time and efficiency
  - How to scale to larger networks?
  - Especially with micro-/nano-second switching times?

# Computation Times

- Trade-off: Computation time and efficiency
  - How to scale to larger networks?
  - Especially with micro-/nano-second switching times?
- Algorithmic idea #1: **Distributed Control Plane**
  - E.g. ProjecToR (Ghobadi et al., 2016) 
  - Next part of the tutorial

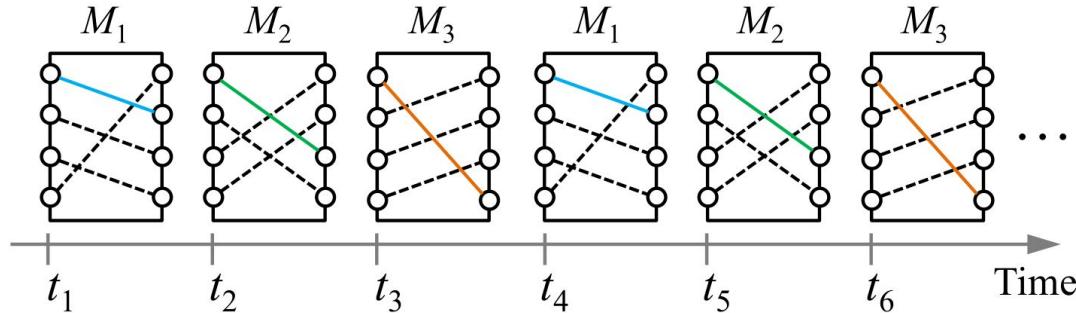
# Computation Times

- Trade-off: Computation time and efficiency
  - How to scale to larger networks?
  - Especially with micro-/nano-second switching times?
- Algorithmic idea #1: **Distributed Control Plane**
  - E.g. ProjecToR (Ghobadi et al., 2016) 
  - Next part of the tutorial
- Algorithmic idea #2: **Oblivious Reconfiguration**
  - Combine topology design and cyclic reconfiguration schedule

# Rotornet

(Mellette et al., SIGCOMM 2017)

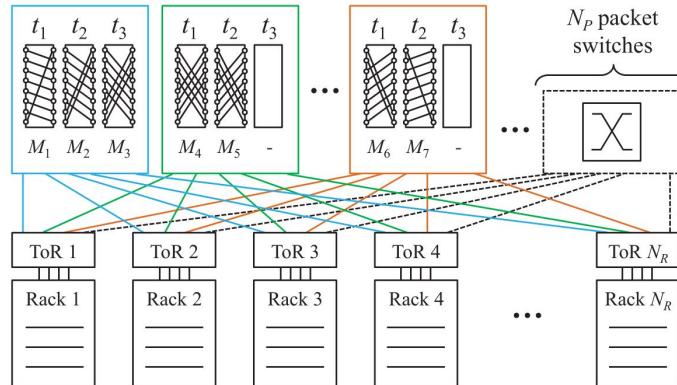
- Cycle through exponentially many matchings?
  - Observe:  $O(n)$  matchings suffice for connectivity
    - Also allows for faster  and cheaper  hardware
- Provision 10%-20% as packet switching for ultra low latency traffic



# Rotornet

(Mellette et al., SIGCOMM 2017)

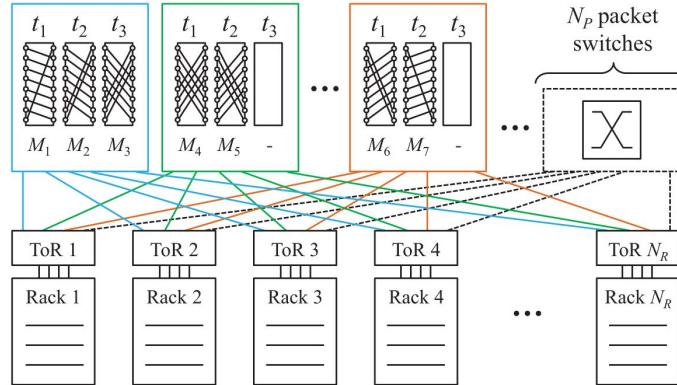
- Parallel less flexible switches to be even faster
  - Distribute matchings over rotor switches
    - E.g. 2048 racks: 16 different matchings with 128 switches
      - Reconfig in microseconds, serve traffic in  $O(1)$ ms
  - Also: No single point of failure



# Rotornet

(Mellette et al., SIGCOMM 2017)

- For uniform traffic: great behaviour with single-hop
- For skewed traffic: leverage Valiant's routing
  - Buffer indirect traffic on per-rack basis
    - Don't hinder direct traffic: distributed offer-accept protocol



# Even More Oblivious: Static Networks

- Observation:
  - Random graphs are great for throughput via multi-hop
    - Build data centers randomly (Singla et al., Jellyfish, NSDI 2012)

# Even More Oblivious: Static Networks

- Observation:
  - Random graphs are great for throughput via multi-hop
    - Build data centers randomly (Singla et al., Jellyfish, NSDI 2012)
- Go deterministic with Expanders: **Beyond fat-trees without antennae, mirrors, and disco-balls**

Algorithmica (2017) 78:1225–1245  
DOI 10.1007/s00453-016-0269-x

## Explicit Expanding Expanders

Michael Dinitz<sup>1</sup> · Michael Schapira<sup>2</sup> ·  
Asaf Valadarsky<sup>2</sup>

Simon Kassing  
ETH Zürich  
simon.kassing@inf.ethz.ch

Asaf Valadarsky  
Hebrew University of Jerusalem  
asaf.valadarsky@mail.huji.ac.il

Gal Shahaf  
Hebrew University of Jerusalem  
gal.shahaf@mail.huji.ac.il

Michael Schapira  
Hebrew University of Jerusalem  
schapiram@mail.huji.ac.il

Ankit Singla  
ETH Zürich  
ankit.singla@inf.ethz.ch

SIGCOMM'17

## Xpander: Towards Optimal-Performance Datacenters

Asaf Valadarsky\*  
asaf.valadarsky@mail.huji.ac.il

Gal Shahaf†  
gal.shahaf@mail.huji.ac.il

Michael Dinitz‡  
mdinitz@cs.jhu.edu

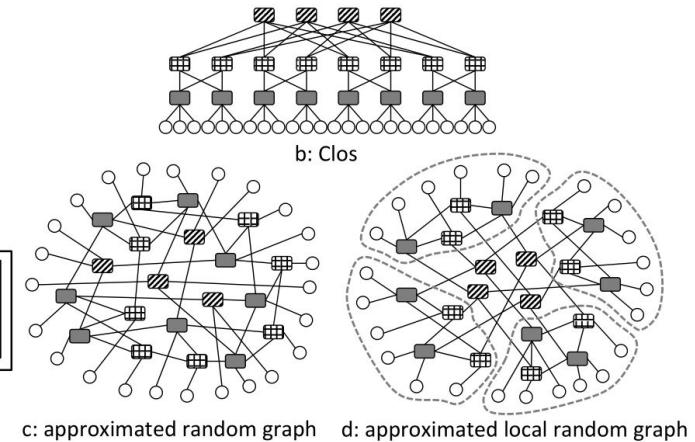
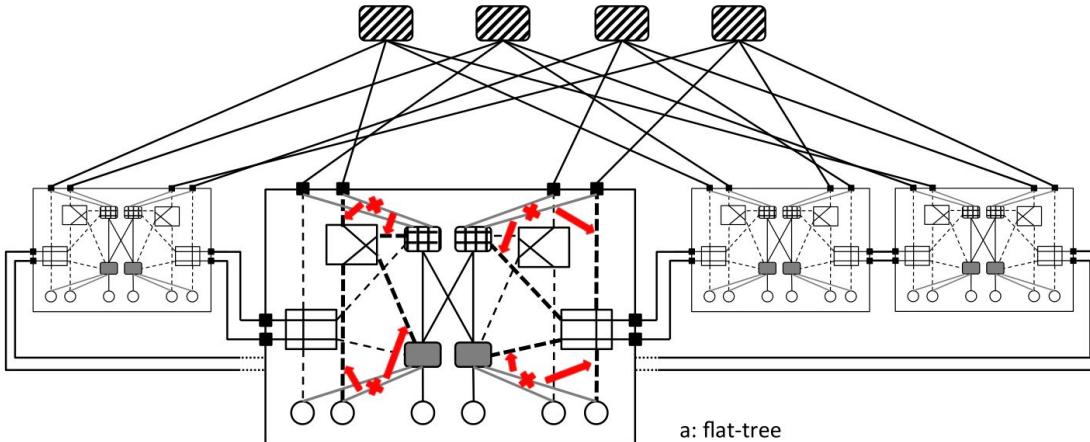
Michael Schapira\*  
schapiram@mail.huji.ac.il

CoNEXT '16

# Flat-Tree

(Xia et al, SIGCOMM 2017)

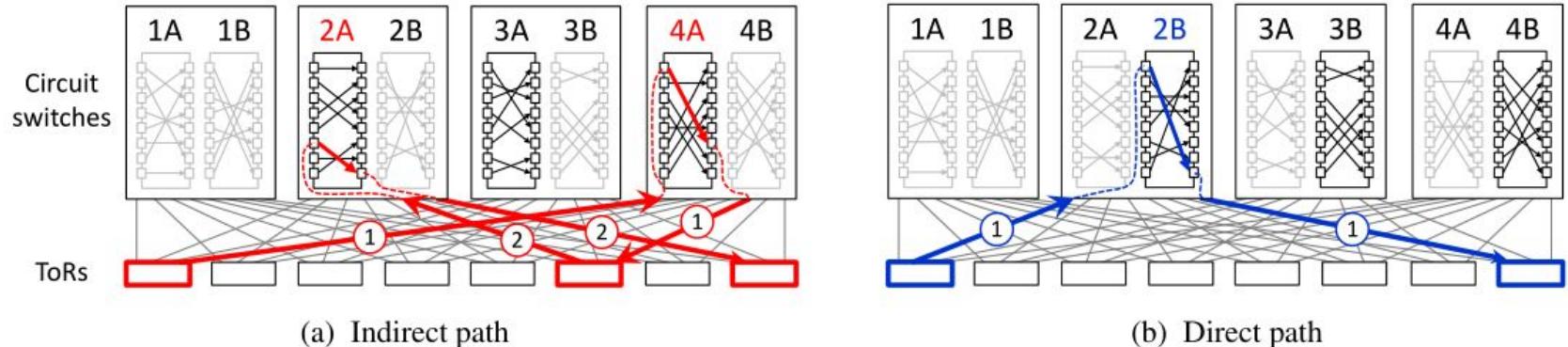
- Flat-Tree (note: not oblivious)
  - Locally convert between random graphs and Clos topologies
  - Use extremely cheap 4/6-port converter switches



# Opera

(Mellette et al, arXiv 2019)

- Extend Rotornet to cycle through Expanders
  - Delay-tolerant traffic can wait for direct connections
  - Other traffic can use the expander networks
    - Low “bandwidth tax”
  - (Valiant 2-hop routing also possible)



# Future (Algorithmic) Work Directions

- Single-hop reconfigurable “XOR” routing is sort of well-understood
  - But mixing with static network parts in general?
- Leveraging multi-hop connections?
  - Efficient heuristics exist, general theoretical framework?
- Speeding up the control plane
  - Oblivious is clearly very fast :-)
  - More distributed approaches

# References

- Long Luo, Klaus-Tycho Foerster, Stefan Schmid, Hongfang Yu: DaRTree: deadline-aware multicast transfers in reconfigurable wide-area networks. IWQoS 2019
- Survey of Reconfigurable Data Center Networks. Foerster and Schmid. SIGACT News, 2019.
- Mohammad Al-Fares, Alexander Loukissas, Amin Vahdat: A scalable, commodity data center network architecture. SIGCOMM 2008
- Arjun Singh et al.: Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. SIGCOMM 2015
- Srikanth Kandula, Jitendra Padhye, Paramvir Bahl: Flyways To De-Congest Data Center Networks. HotNets 2009
- Manya Ghobadi et al.: ProjecToR: Agile Reconfigurable Data Center Interconnect. SIGCOMM 2016
- Daniel Halperin, Srikanth Kandula, Jitendra Padhye, Paramvir Bahl, David Wetherall: Augmenting data center networks with multi-gigabit wireless links. SIGCOMM 2011
- Nathan Farrington et al.: Helios: a hybrid electrical/optical switch architecture for modular data centers. SIGCOMM 2010
- Guohui Wang et al.: c-Through: part-time optics in data centers. SIGCOMM 2010
- Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang: Proteus: a topology malleable data center network. HotNets 2010
- Kai Chen et al.: OSA: An Optical Switching Architecture for Data Center Networks With Unprecedented Flexibility. IEEE/ACM Trans. Netw. 22(2): 498-511 (2014)
- Ankit Singla, Atul Singh, Yan Chen: OSA: An Optical Switching Architecture for Data Center Networks with Unprecedented Flexibility. NSDI 2012
- Navid Hamed Azimi et al.: FireFly: a reconfigurable wireless data center fabric using free-space optics. SIGCOMM 2014
- Yiting Xia et al.: A Tale of Two Topologies: Exploring Convertible Data Center Network Architectures with Flat-tree. SIGCOMM 2017
- Xia Zhou et al.: Mirror mirror on the ceiling: flexible wireless links for data centers. SIGCOMM 2012
- Klaus-Tycho Foerster, Manya Ghobadi, Stefan Schmid: Characterizing the algorithmic complexity of reconfigurable data center architectures. ANCS 2018
- T. Fenz, K.-T. Foerster, S. Schmid, A. Villedieu: Efficient Non-Segregated Routing for Reconfigurable Demand-Aware Networks. IFIP Networking 2019
- K.-T. Foerster et al.: On the Complexity of Non-Segregated Routing in Reconfigurable Data Center Architectures. ACM SIGCOMM CCR 49(2): 3-8 (2019)
- George Porter et al: Integrating microsecond circuit switching into the data center. SIGCOMM 2013
- Li Chen et al.: Enabling Wide-Spread Communications on Optical Fabric with MegaSwitch. NSDI 2017
- William M. Mellette et al.: RotorNet: A Scalable, Low-complexity, Optical Datacenter Network. SIGCOMM 2017
- He Liu et al.: Scheduling techniques for hybrid circuit/packet networks. CoNEXT 2015
- Shaileshh Bojja Venkatakrishnan et al.:Costly circuits, submodular schedules and approximate Carathéodory Theorems. SIGMETRICS 2016
- Shaileshh Bojja Venkatakrishnan et al.: Costly circuits, submodular schedules and approximate Carathéodory Theorems. Queueing Syst. 88(3-4): 311-347 (2018)
- William M. Mellette et al.: Expanding across time to deliver bandwidth efficiency and low latency. arXiv:1903.12307

# Reconfigurable Networks: Enablers, Algorithms, Complexity

Ramakrishnan Durairajan, Klaus-Tycho Forster, Stefan Schmid

Tutorial @ ACM Sigmetrics 2019  
Phoenix, Arizona, USA

# Reconfigurable Networks: Enablers, Algorithms, Complexity

Ramakrishnan Durairajan, Klaus-Tycho Förster, Stefan Schmid

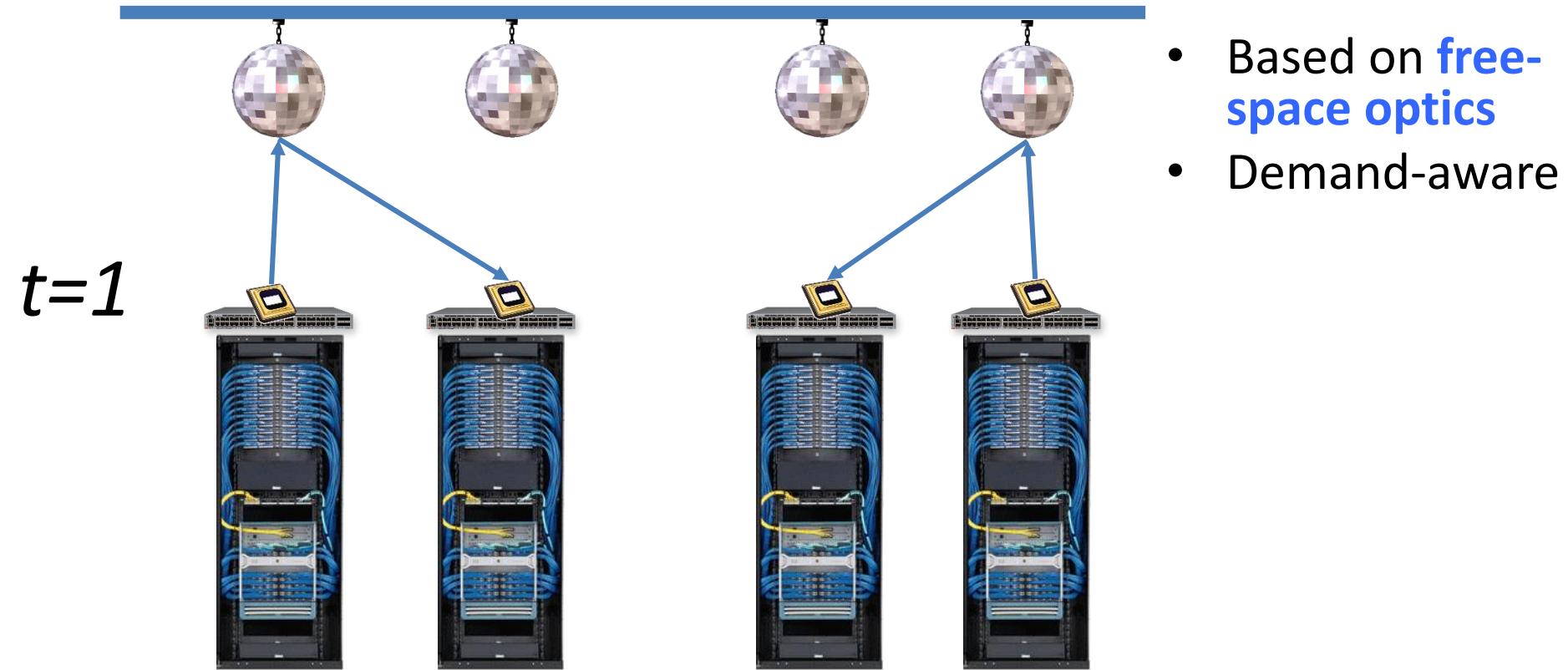
Tutorial @ ACM Sigmetrics 2019  
Phoenix, Arizona, USA

# A demand-aware network with fast reconfigurations + high fanout: ProjecToR

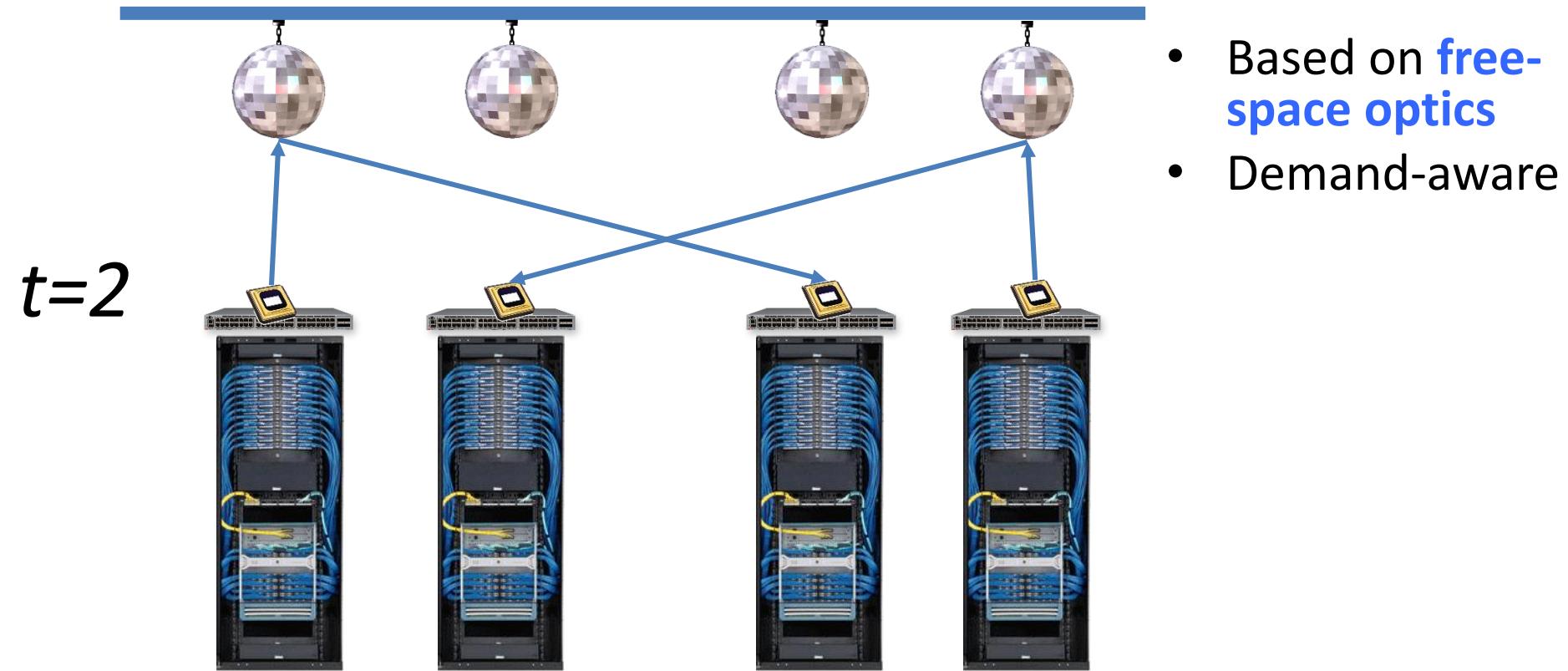


Manya Ghobadi et al.\*  
(\* collaborators from U. Arizona!)  
*Kudos for some slides!*

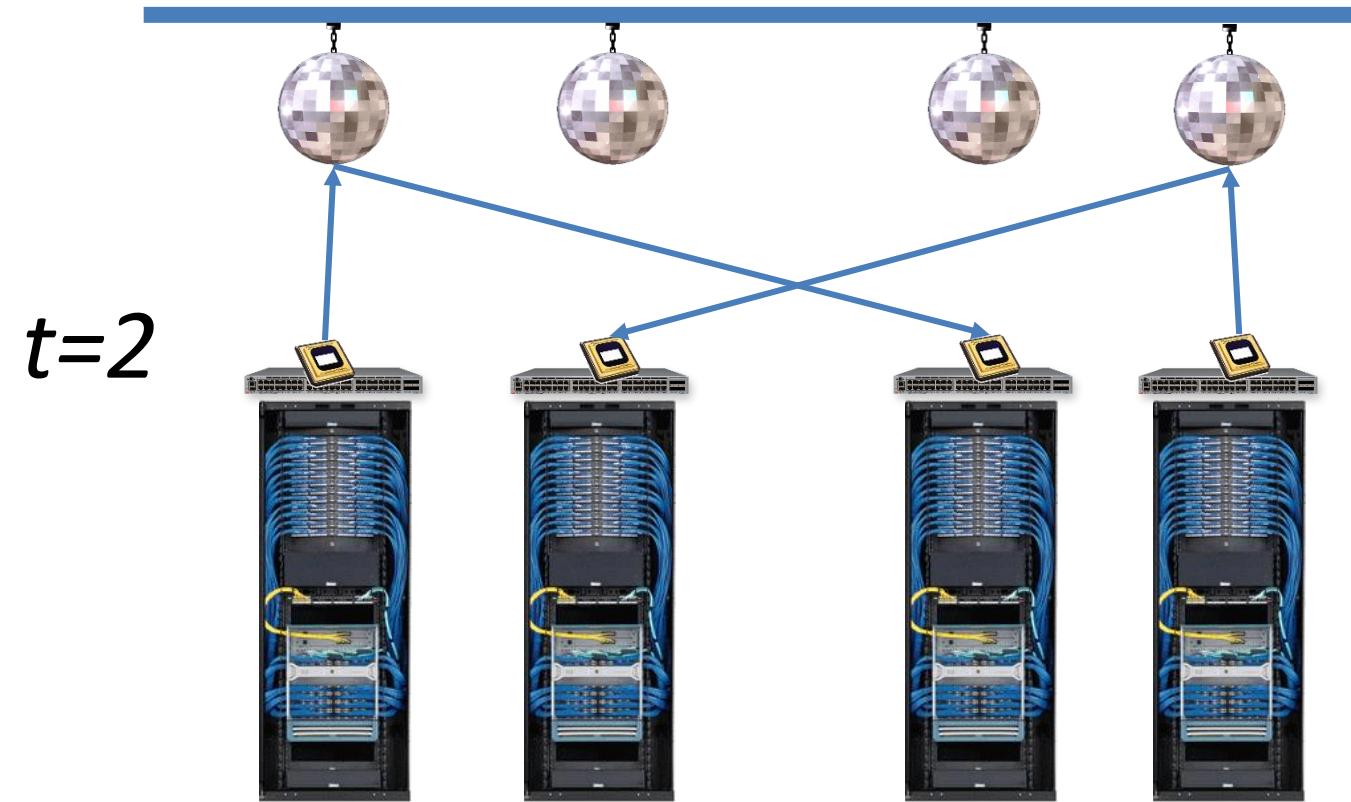
# ProjectToR



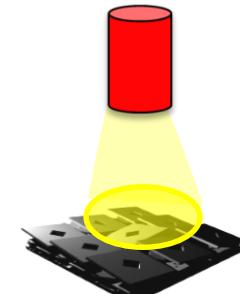
# ProjectToR



# ProjectToR

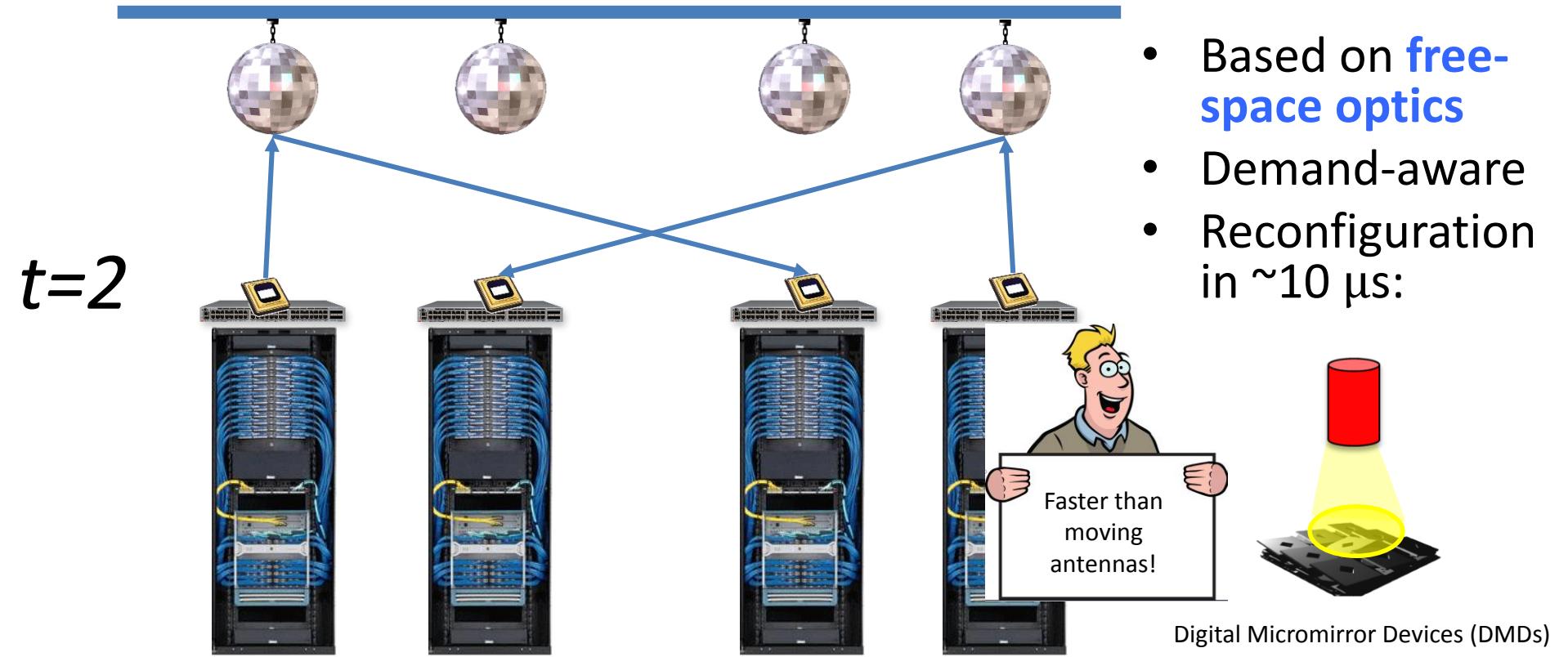


- Based on **free-space optics**
- Demand-aware
- Reconfiguration in  $\sim 10 \mu\text{s}$ :



Digital Micromirror Devices (DMDs)

# ProjectToR



# ProjecToR in More Details: Empirical Motivation

|   | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 3 | 3 | 3 |
| B | 3 | 0 | 3 | 3 |
| C | 3 | 3 | 0 | 3 |
| D | 3 | 3 | 3 | 0 |

|   | A | B  | C | D |
|---|---|----|---|---|
| A | 0 | 6  | 6 | 0 |
| B | 0 | 0  | 0 | 0 |
| C | 0 | 0  | 0 | 0 |
| D | 0 | 12 | 8 | 0 |

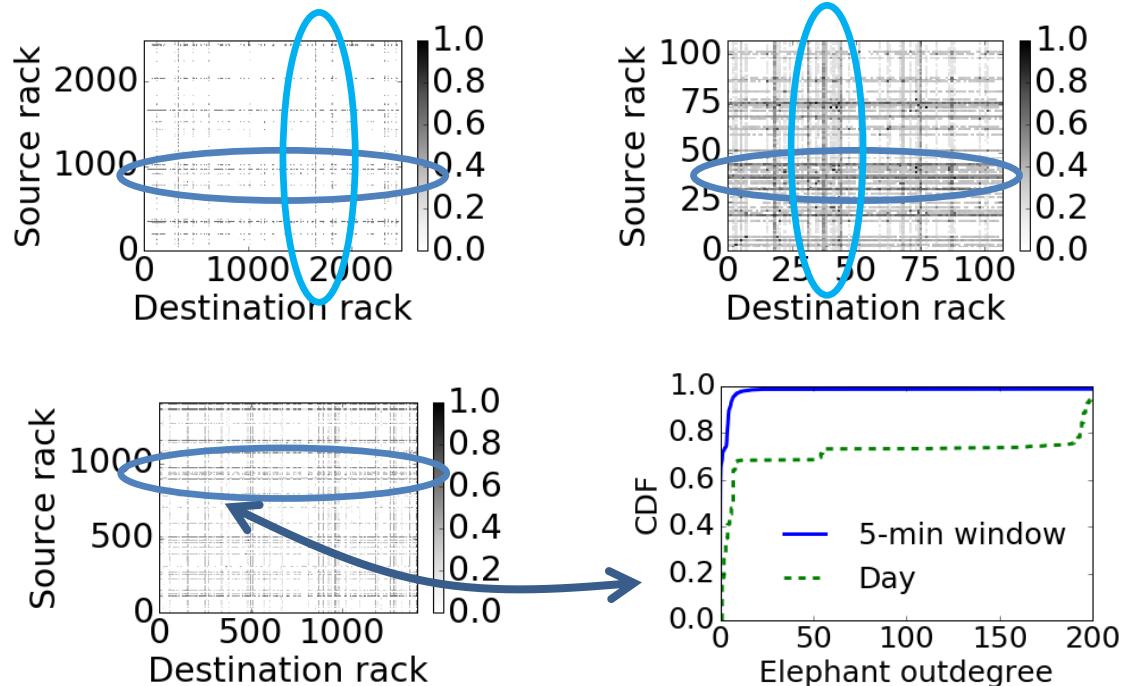
Ideal demand matrix:  
uniform and static

Non-ideal demand matrix:  
**skewed** and **dynamic**

# ProjectToR in More Details: Empirical Motivation

## Observation 1:

- Many rack pairs exchange **little traffic**
- Only some **hot rack pairs** are active



## Observation 2:

- Some source racks send large amounts of traffic **to many other racks**

Microsoft data: 200K servers across 4 production clusters, cluster sizes: 100 - 2500 racks.  
Mix of workloads: MapReduce-type jobs, index builders, database and storage systems.

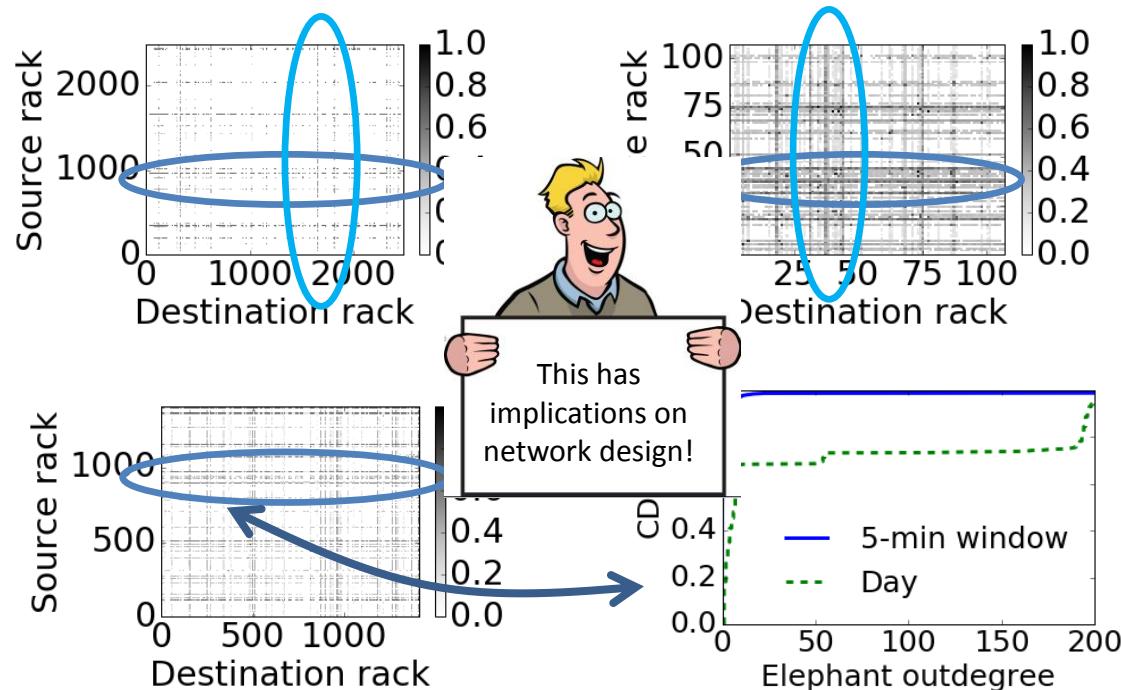
# ProjectToR in More Details: Empirical Motivation

## Observation 1:

- Many rack pairs exchange **little traffic**
- Only some **hot rack pairs** are active

## Observation 2:

- Some source racks send large amounts of traffic **to many other racks**

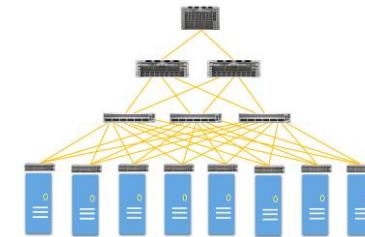


Microsoft data: 200K servers across 4 production clusters, cluster sizes: 100 - 2500 racks.  
Mix of workloads: MapReduce-type jobs, index builders, database and storage systems.

# ProjecToR in More Details: Empirical Motivation

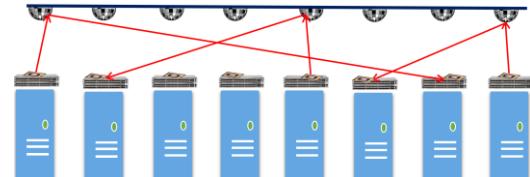
Implication for **static topology** with uniform capacity:

- **Over-provisioned** for most rack pairs
- **Under-provisioned** for few others

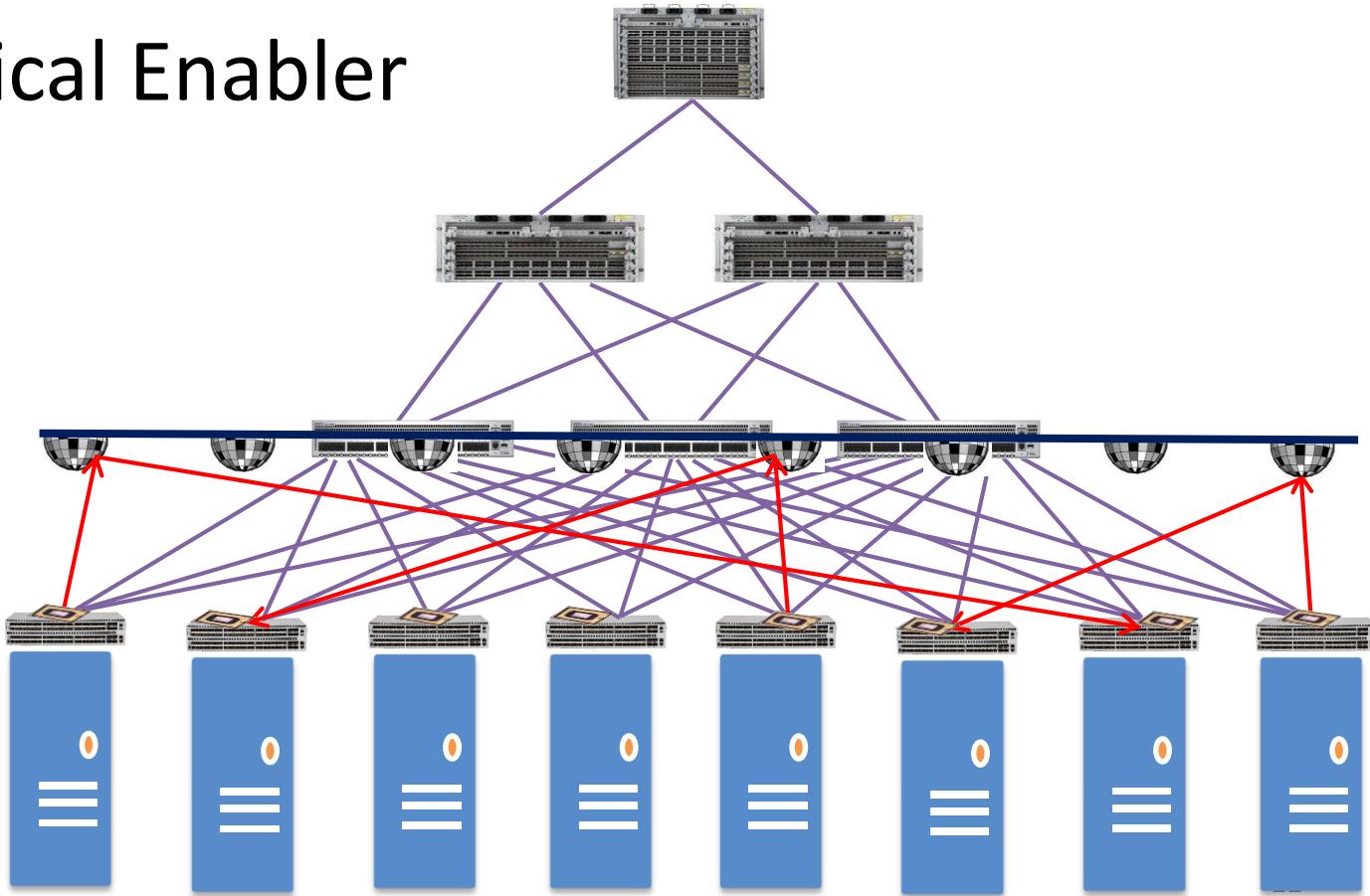
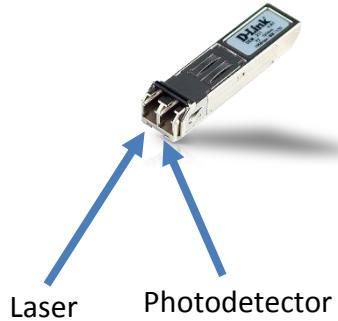


Implications for **dynamic topology**:

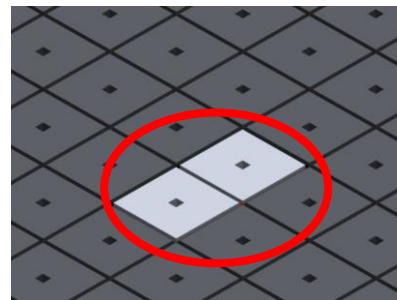
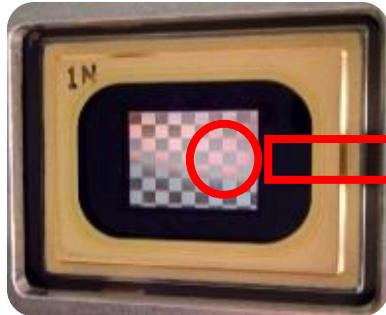
- Must be able to create direct links to lots of other racks (**high fan-out**)
- And switch quickly among destinations (**low switching time**)



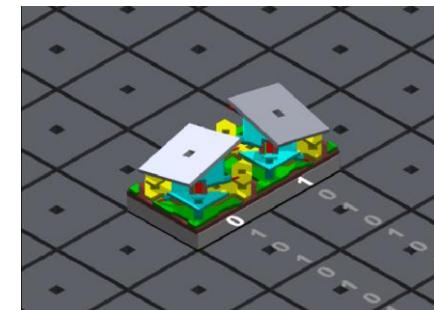
# ProjectToR in More Details: Technological Enabler



# ProjecToR in More Details: DMDs



Array of  
micromirrors

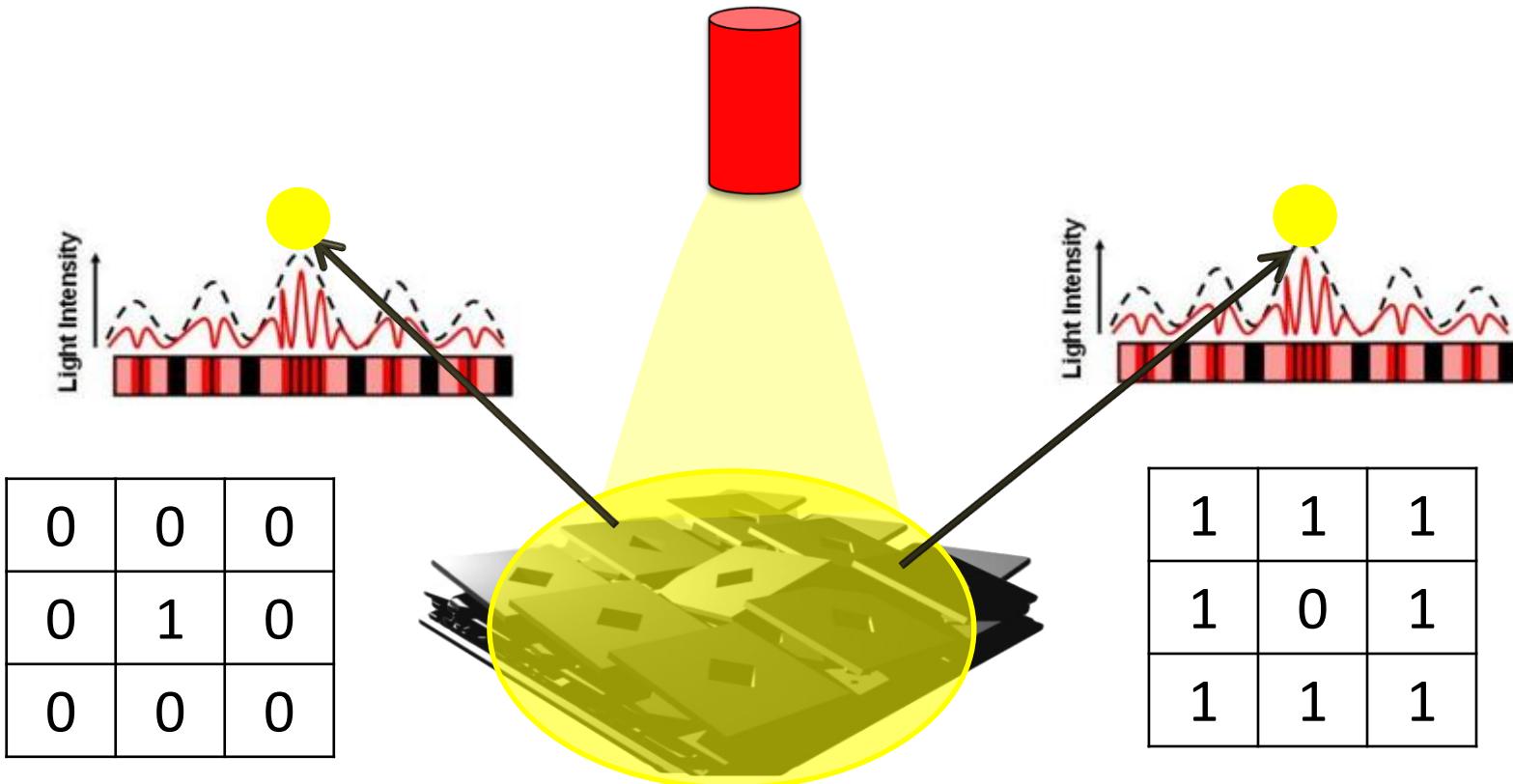


Memory cell

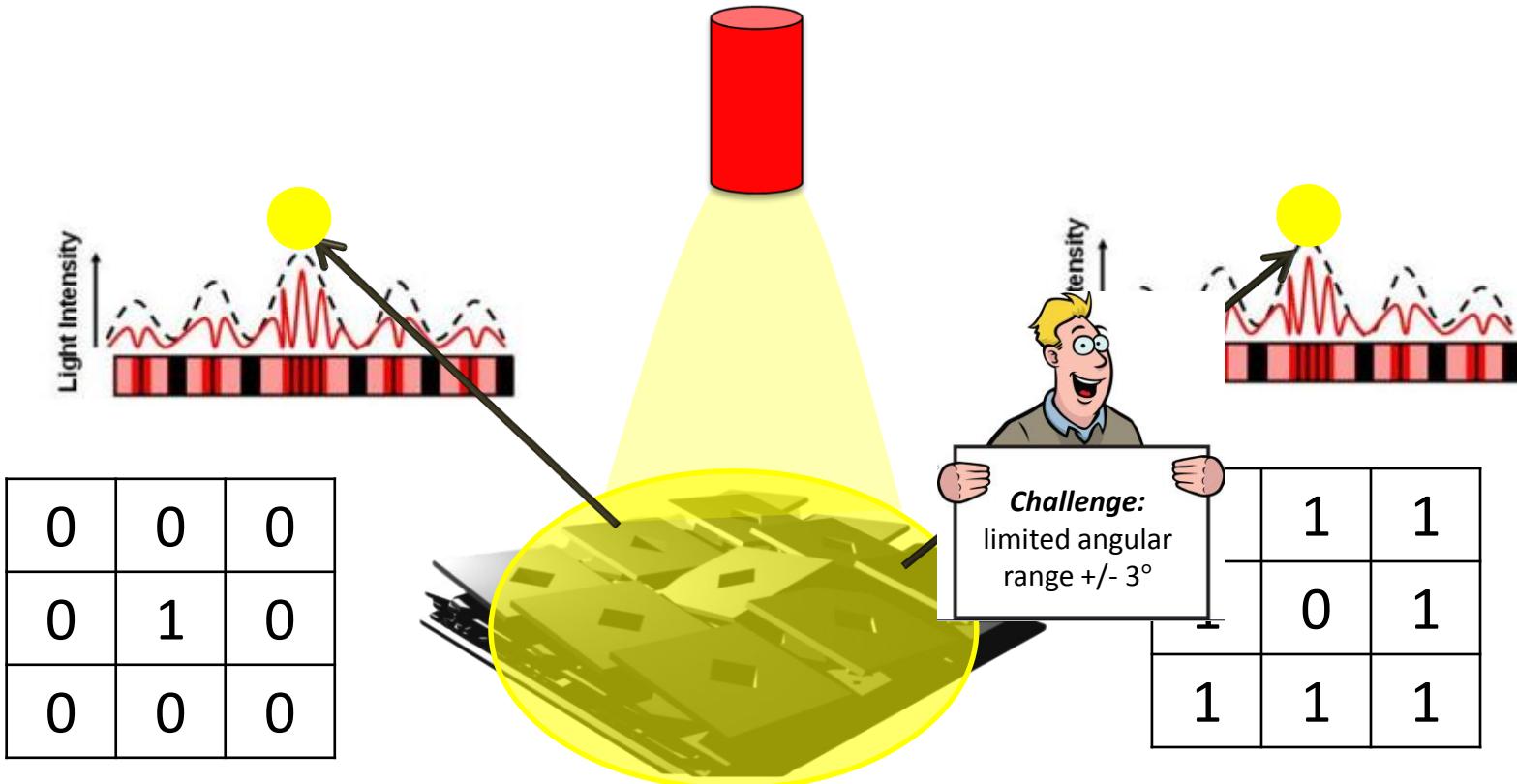


- Each micromirror can be turned on/off
- Essentially a **0/1-image**: e.g., array size 768 x 1024
- Direction of the diffracted light can be finely tuned

# ProjectToR in More Details: DMDs to Redirect Light *Fast*



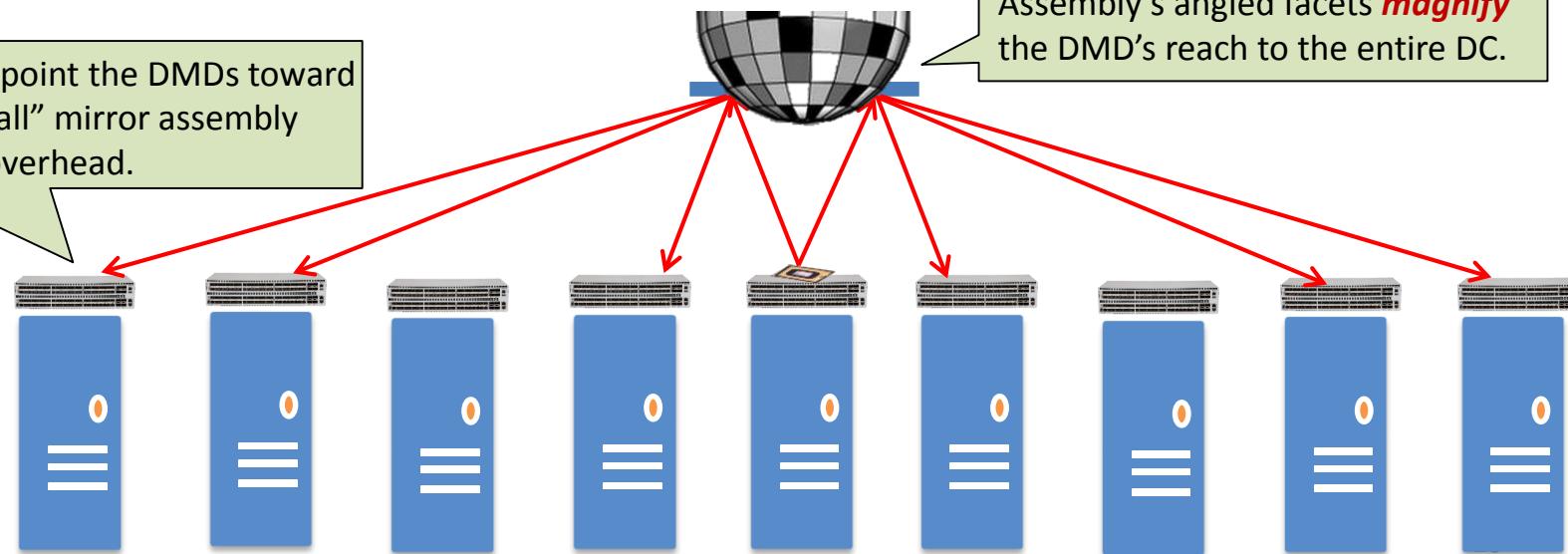
# ProjectToR in More Details: DMDs to Redirect Light *Fast*



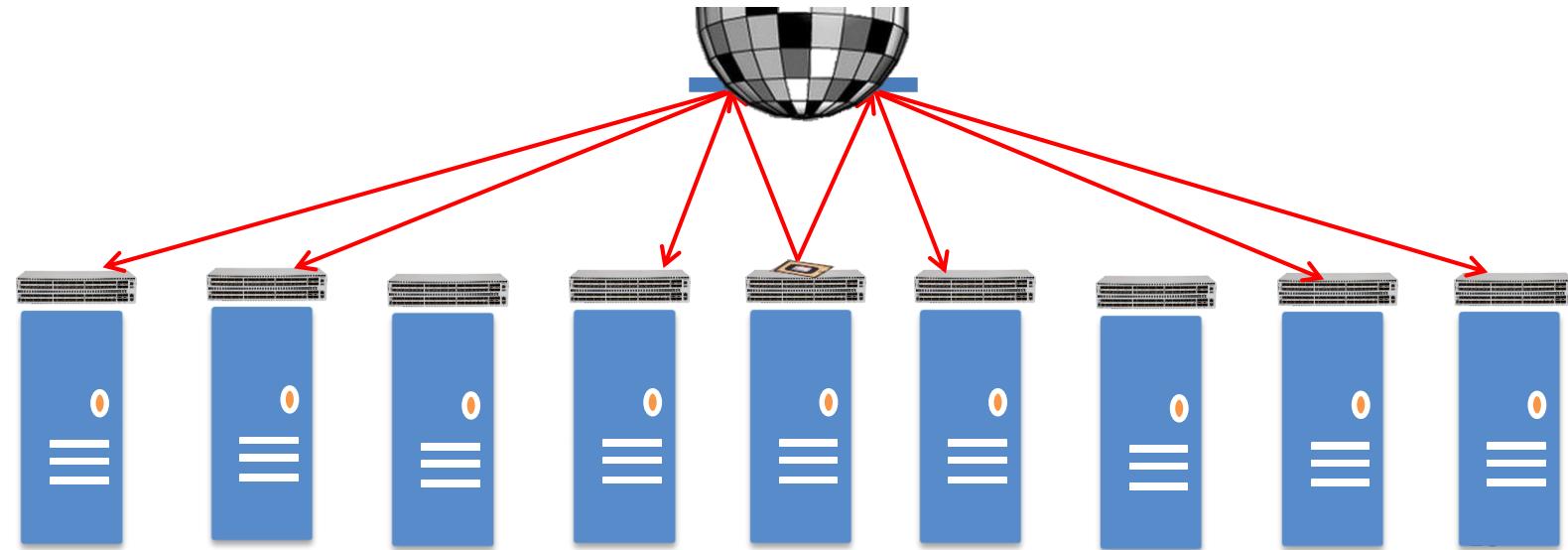
# ProjectToR in More Details: Coupling DMDs with angled mirrors

**Coupling:** point the DMDs toward a “disco-ball” mirror assembly installed overhead.

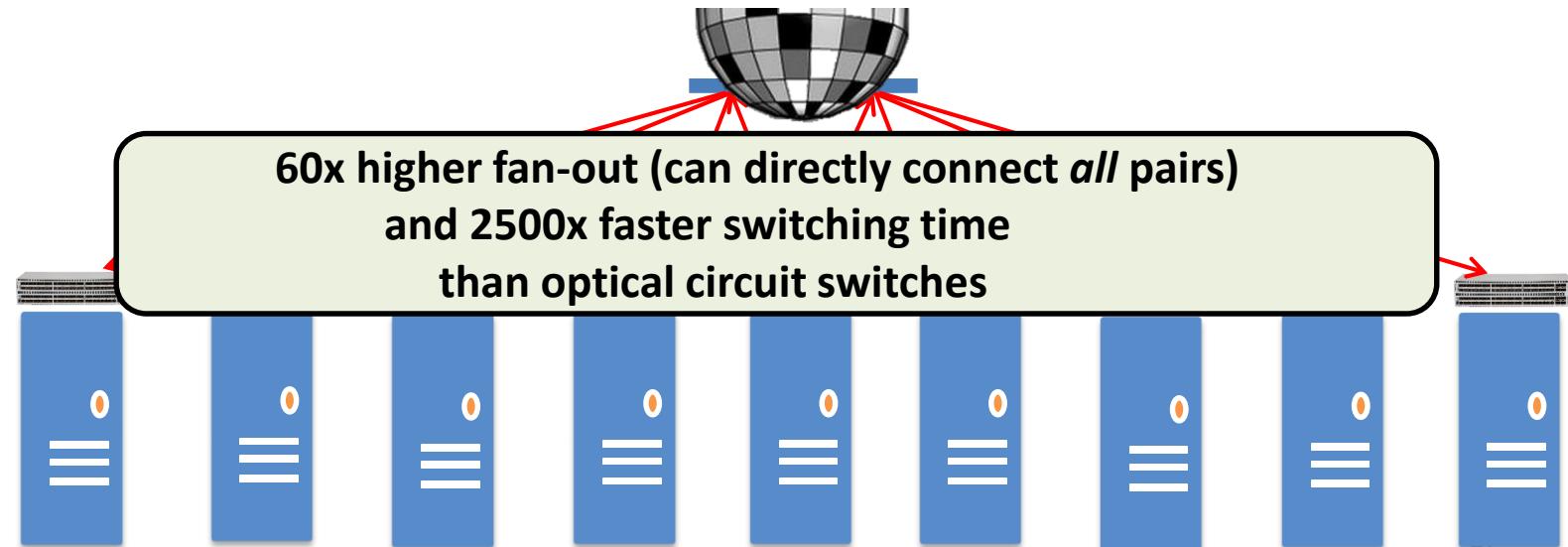
Assembly's angled facets **magnify** the DMD's reach to the entire DC.



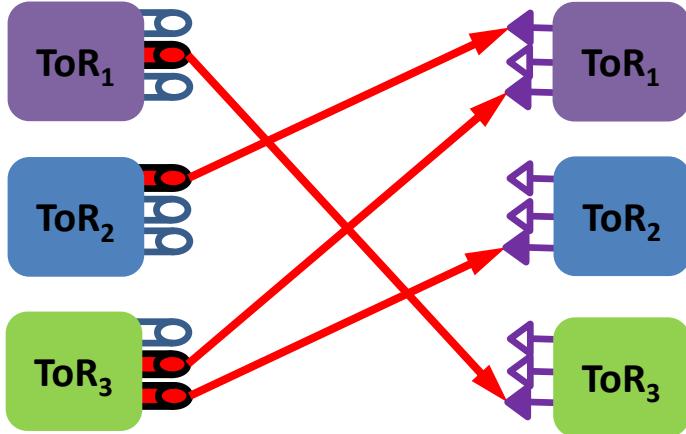
# ProjectToR in More Details: Coupling DMDs with angled mirrors



# ProjecToR in More Details: Coupling DMDs with angled mirrors



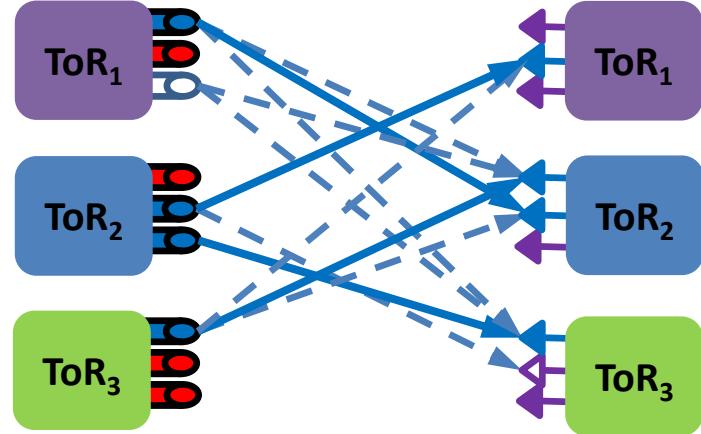
# ProjectToR in More Details: 2-Topology Approach



dedicated topology  
(multihop, changes slowly)

k-shortest paths routing

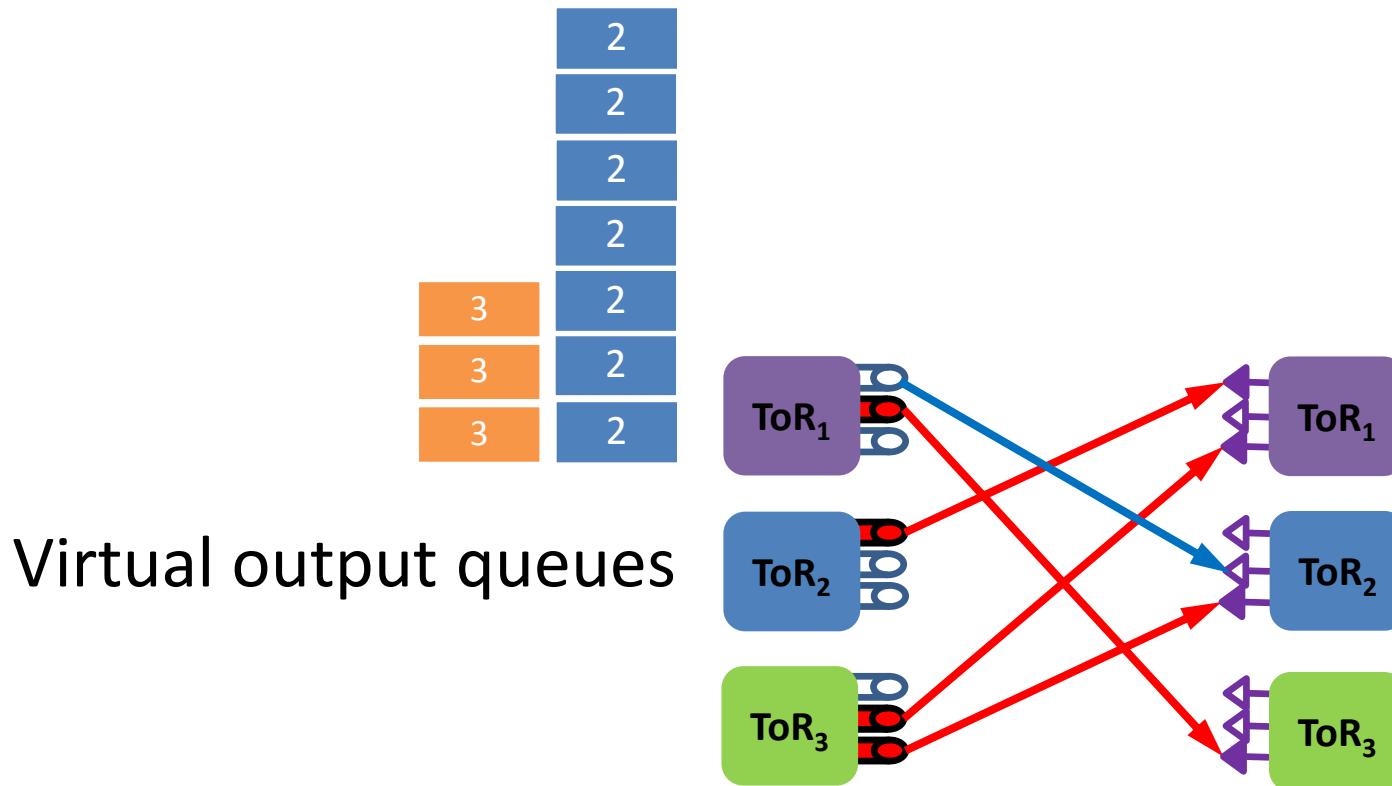
+



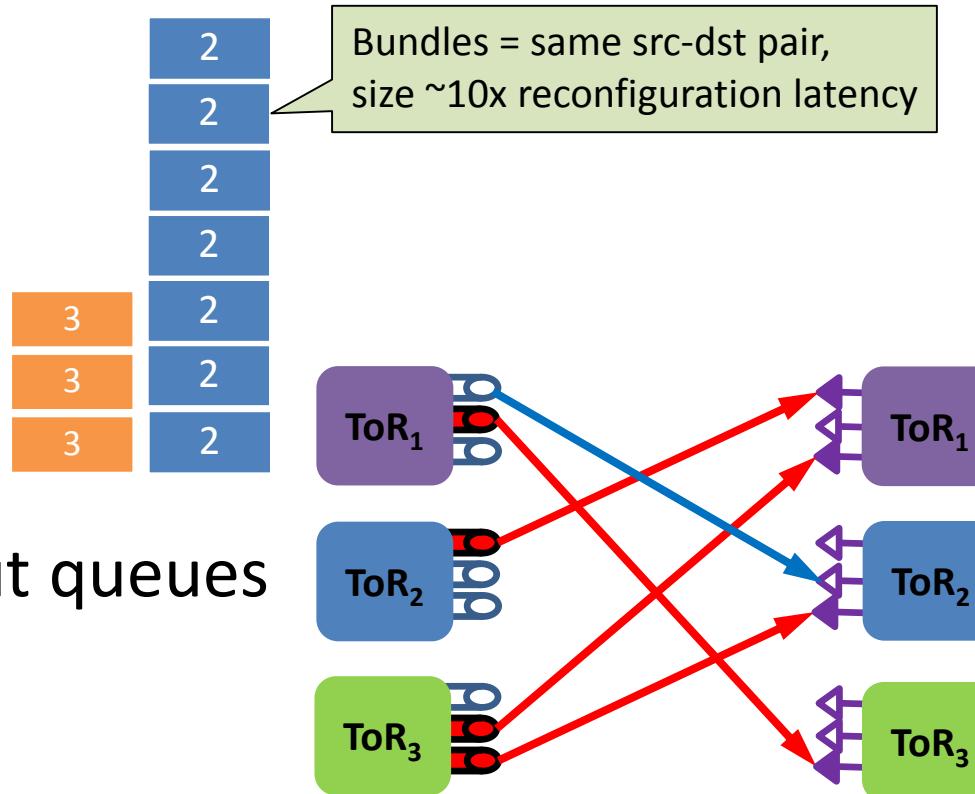
opportunistic links  
(singlehop, changes fast)

Decentralized stable matching

# ProjectToR in More Details: Based on Virtual Output Queue Technique

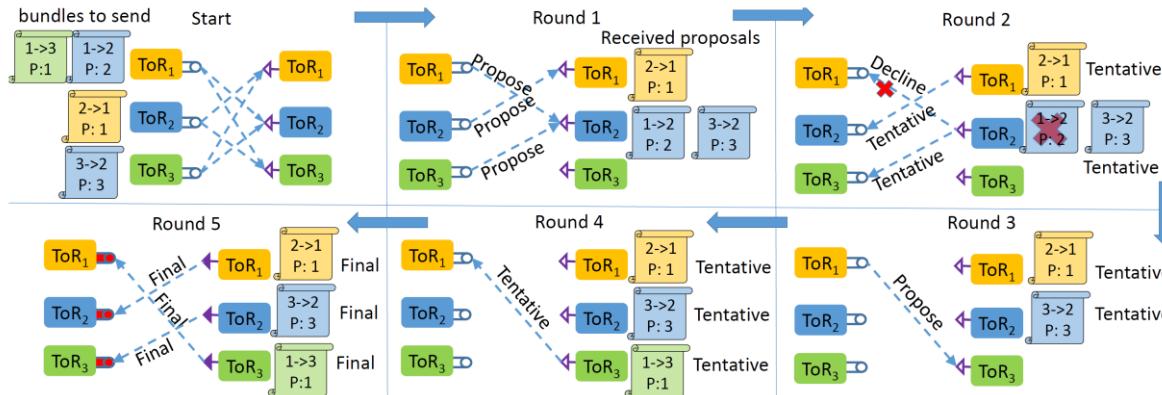


# ProjectToR in More Details: Based on Virtual Output Queue Technique



# ProjectToR in More Details: Matching with a Twist

- Challenge: scheduling problem is **2-tiered**:
  - While **traffic matrix** is between ToRs
  - matching** occurs between lasers and photodetectors: multiples of those per ToR!



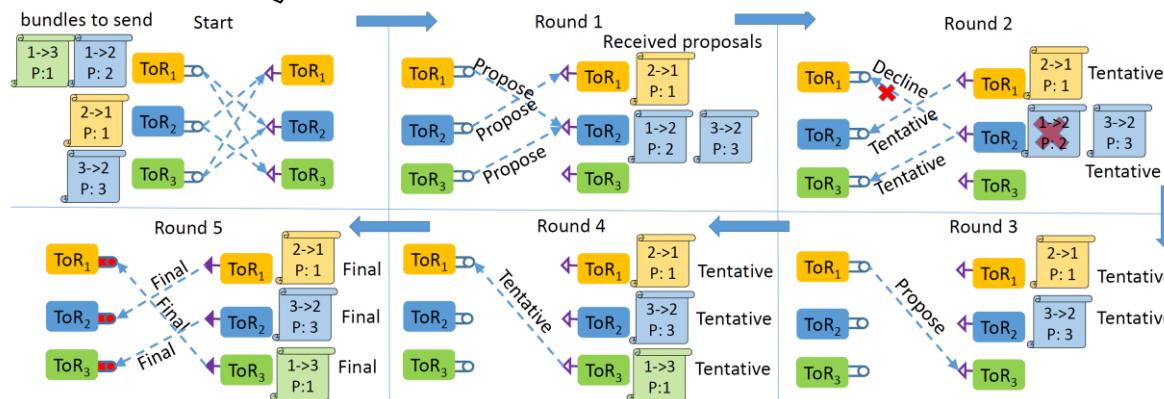
# ProjectToR in More Details: Matching with a Twist

- Challenge: scheduling problem is **2-tiered**:

While ~~traffic matrix~~ is between ToRs

Each ToR has an *interest map*:  
outstanding bundles and their priorities

Users and photodetectors: multiples of those per ToR!



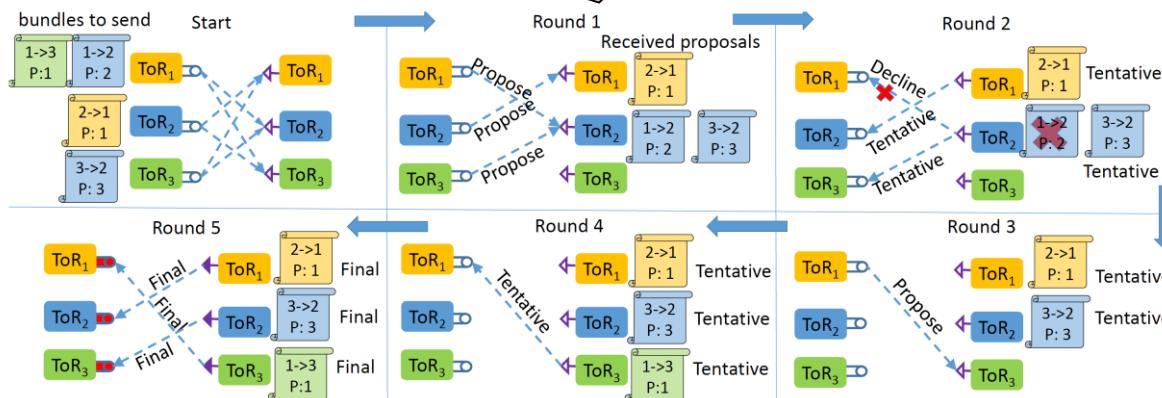
# ProjectToR in More Details: Matching with a Twist

- Challenge: scheduling problem is **2-tiered**:

While traffic matrix is between ToRs

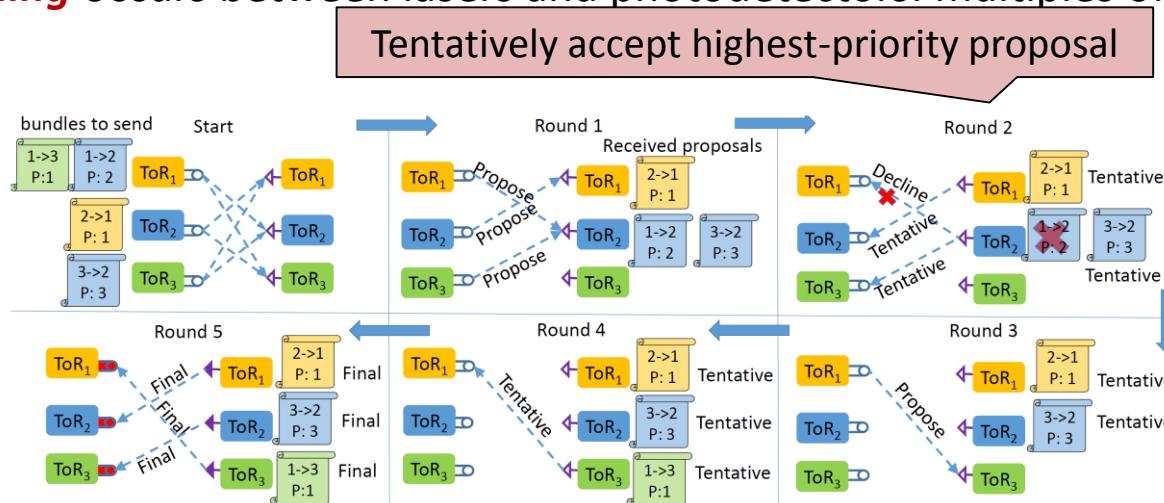
All ToRs **send a proposal** for their top priority bundle to the corresponding destination

ToRs: multiples of those per ToR!



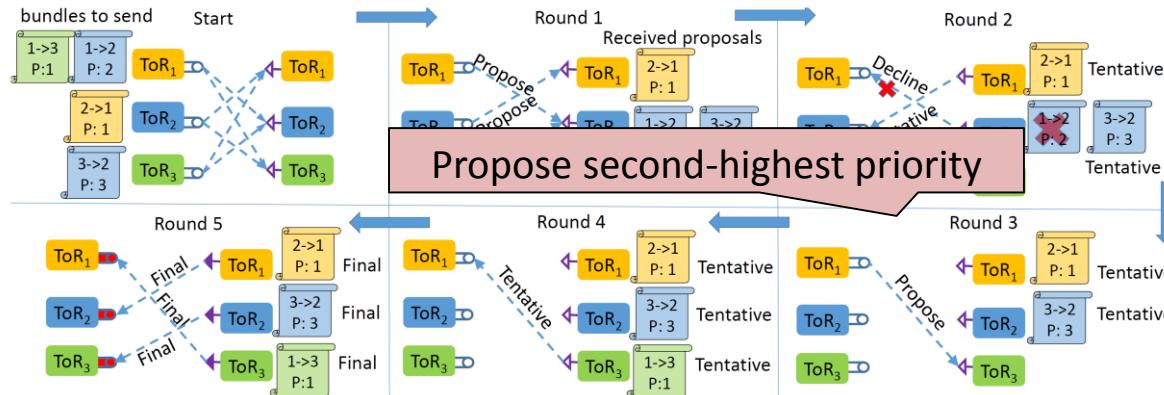
# ProjectToR in More Details: Matching with a Twist

- Challenge: scheduling problem is **2-tiered**:
  - While **traffic matrix** is between ToRs
  - matching** occurs between lasers and photodetectors: multiples of those per ToR!



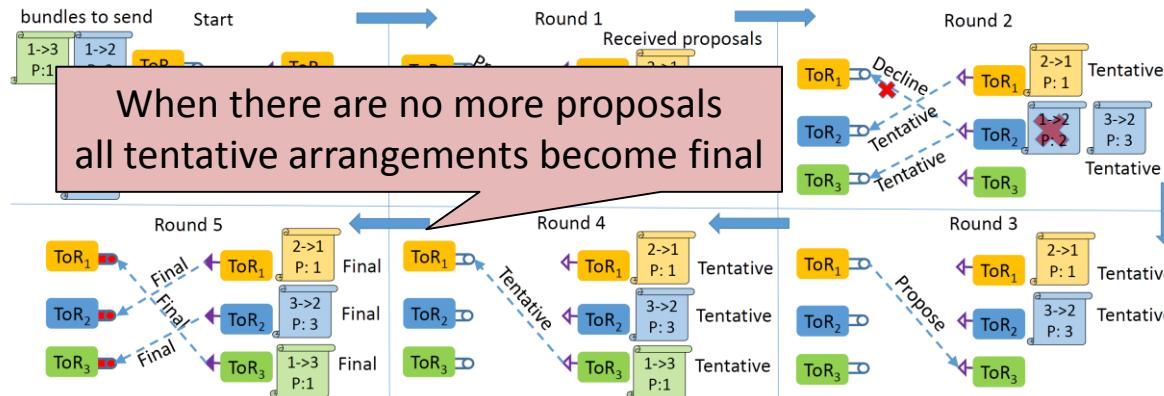
# ProjectToR in More Details: Matching with a Twist

- Challenge: scheduling problem is **2-tiered**:
  - While **traffic matrix** is between ToRs
  - matching** occurs between lasers and photodetectors: multiples of those per ToR!



# ProjectToR in More Details: Matching *with a Twist*

- Challenge: scheduling problem is **2-tiered**:
  - While **traffic matrix** is between ToRs
  - matching** occurs between lasers and photodetectors: multiples of those per ToR!





---

Such Fast Reconfigurability Enables  
Demand-Aware Networks (DANs)!

---

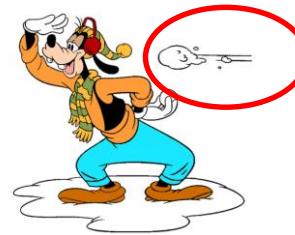
**You:** I invented a great new reconfigurable network which allows to self-adjust to the demand it serves!

**Boss:** Okay, so how much better is your demand-aware network really compared to demand-oblivious networks!?

**You:** hmm...

# A Simple Answer

Demand-Oblivious Networks =



# The *SIGMETRICS* Answer

- It depends...

# The *SIGMETRICS* Answer

As always in  
computer  
science! ☺

- It depends...
- ... on the demand!



We need **metrics!**

# Roadmap

- Entropy: A metric for demand-aware networks?
  - Intuition
  - A lower bound
  - Algorithms achieving entropy bounds
- From static to dynamic demand-aware networks
  - Empirical motivation
  - A connection to self-adjusting datastructures



# Roadmap

- Entropy: A metric for demand-aware networks?
  - Intuition
  - A lower bound
  - Algorithms achieving entropy bounds
- From static to dynamic demand-aware networks
  - Empirical motivation
  - A connection to self-adjusting datastructures



---

# A Simple Example

---

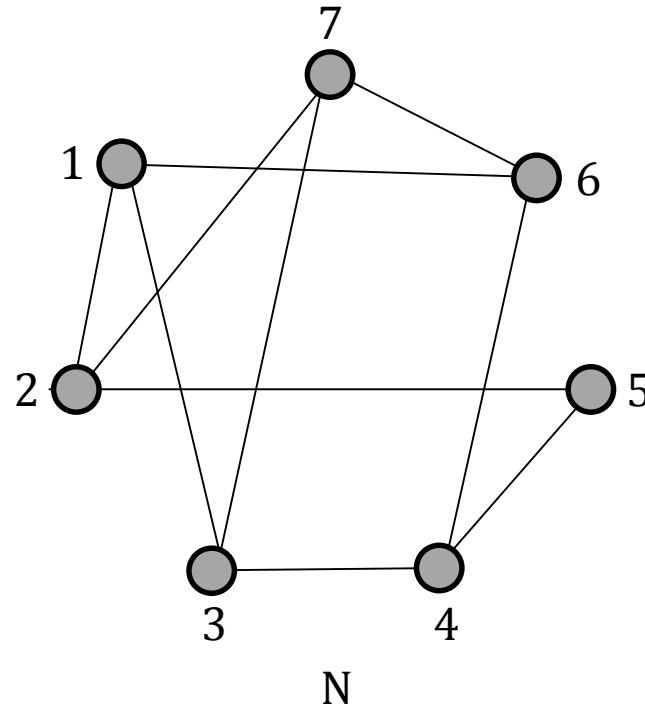
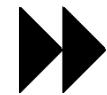
# Input: Workload

Destinations

|         |   | Destinations   |                |                |                |                |                |                |
|---------|---|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
|         |   | 1              | 2              | 3              | 4              | 5              | 6              | 7              |
| Sources | 1 | 0              | $\frac{2}{65}$ | $\frac{1}{13}$ | $\frac{1}{65}$ | $\frac{1}{65}$ | $\frac{2}{65}$ | $\frac{3}{65}$ |
|         | 2 | $\frac{2}{65}$ | 0              | $\frac{1}{65}$ | 0              | 0              | 0              | $\frac{2}{65}$ |
|         | 3 | $\frac{1}{13}$ | $\frac{1}{65}$ | 0              | $\frac{2}{65}$ | 0              | 0              | $\frac{1}{13}$ |
|         | 4 | $\frac{1}{65}$ | 0              | $\frac{2}{65}$ | 0              | $\frac{4}{65}$ | 0              | 0              |
|         | 5 | $\frac{1}{65}$ | 0              | $\frac{3}{65}$ | $\frac{4}{65}$ | 0              | 0              | 0              |
|         | 6 | $\frac{2}{65}$ | 0              | 0              | 0              | 0              | 0              | $\frac{3}{65}$ |
|         | 7 | $\frac{3}{65}$ | $\frac{2}{65}$ | $\frac{1}{13}$ | 0              | 0              | $\frac{3}{65}$ | 0              |

$\mathcal{D}$

# Output: Constant-Degree DAN



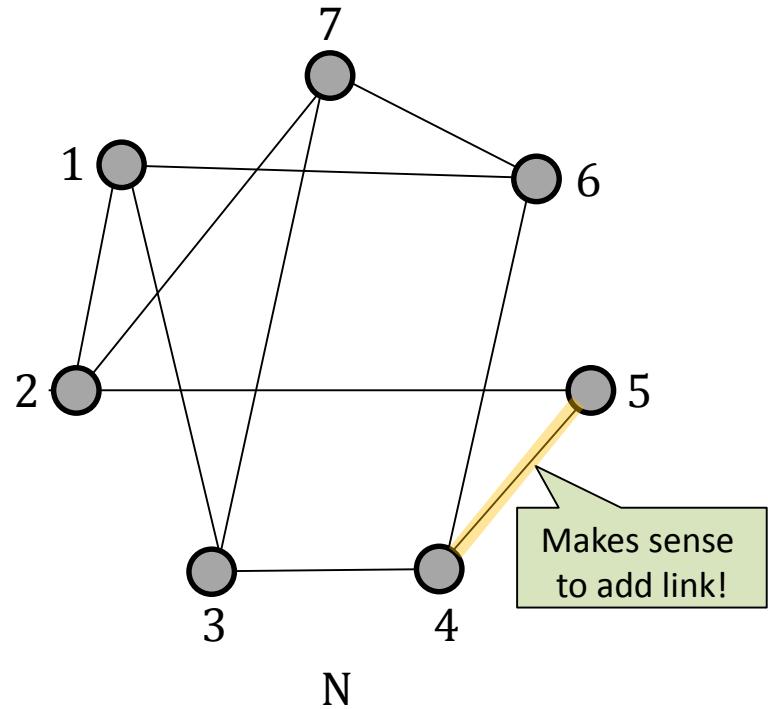
# Input: Workload

Destinations

|   | 1              | 2              | 3              | 4              | 5                 | 6              | 7              |
|---|----------------|----------------|----------------|----------------|-------------------|----------------|----------------|
| 1 | 0              | $\frac{2}{65}$ | $\frac{1}{13}$ | $\frac{1}{65}$ | $\frac{1}{65}$    | $\frac{2}{65}$ | $\frac{3}{65}$ |
| 2 | $\frac{2}{65}$ | 0              | $\frac{1}{65}$ | 0              | 0                 | 0              | $\frac{2}{65}$ |
| 3 | $\frac{1}{13}$ | $\frac{1}{65}$ | 0              | $\frac{2}{65}$ | Much from 4 to 5. |                |                |
| 4 | $\frac{1}{65}$ | 0              | $\frac{2}{65}$ | 0              | $\frac{4}{65}$    | 0              | 0              |
| 5 | $\frac{1}{65}$ | 0              | $\frac{3}{65}$ | $\frac{4}{65}$ | 0                 | 0              | 0              |
| 6 | $\frac{2}{65}$ | 0              | 0              | 0              | 0                 | $\frac{3}{65}$ |                |
| 7 | $\frac{3}{65}$ | $\frac{2}{65}$ | $\frac{1}{13}$ | 0              | 0                 | $\frac{3}{65}$ | 0              |

$\mathcal{D}$

# Output: Constant-Degree DAN



# Input: Workload

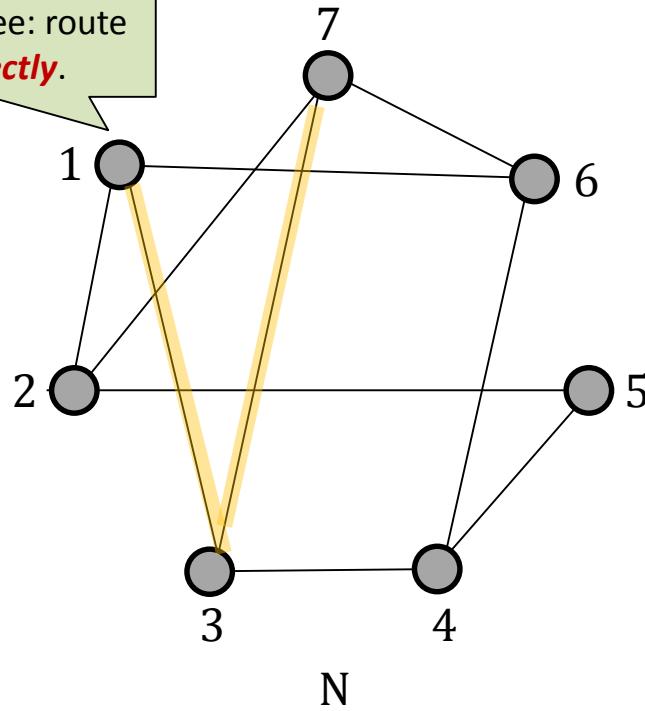
# Output: Constant-Degree DAN

1 communicates to many.

|         |   | Destinations   |                |                |                |                |                |                |
|---------|---|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
|         |   | 2              | 3              | 4              | 5              | 6              | 7              |                |
| Sources | 1 | 0              | $\frac{2}{65}$ | $\frac{1}{13}$ | $\frac{1}{65}$ | $\frac{1}{65}$ | $\frac{2}{65}$ | $\frac{3}{65}$ |
|         | 2 | $\frac{2}{65}$ | 0              | $\frac{1}{65}$ | 0              | 0              | 0              | $\frac{2}{65}$ |
|         | 3 | $\frac{1}{13}$ | $\frac{1}{65}$ | 0              | $\frac{2}{65}$ | 0              | 0              | $\frac{1}{13}$ |
|         | 4 | $\frac{1}{65}$ | 0              | $\frac{2}{65}$ | 0              | $\frac{4}{65}$ | 0              | 0              |
|         | 5 | $\frac{1}{65}$ | 0              | $\frac{3}{65}$ | $\frac{4}{65}$ | 0              | 0              | 0              |
|         | 6 | $\frac{2}{65}$ | 0              | 0              | 0              | 0              | $\frac{3}{65}$ |                |
|         | 7 | $\frac{3}{65}$ | $\frac{2}{65}$ | $\frac{1}{13}$ | 0              | 0              | $\frac{3}{65}$ | 0              |

$\mathcal{D}$

Bounded degree: route to 7 *indirectly*.



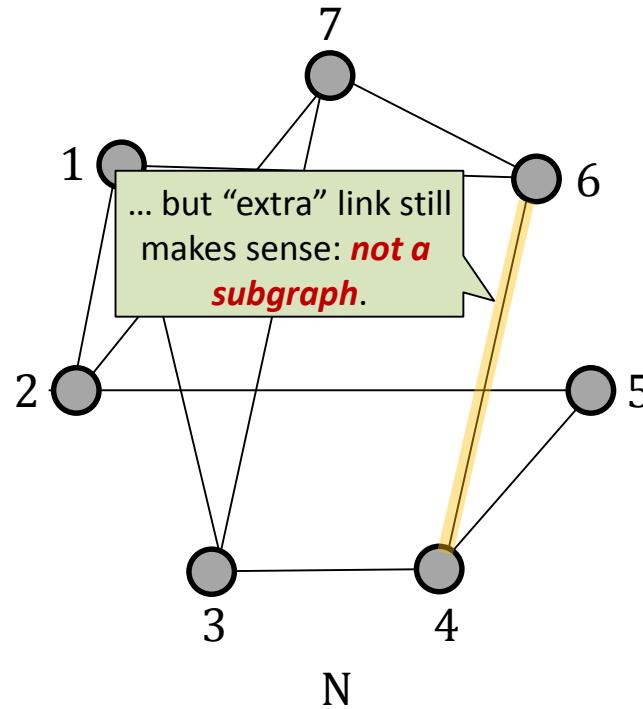
# Input: Workload

Destinations

|   | 1              | 2              | 3              | 4              | 5              | 6              | 7              |
|---|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 1 | 0              | $\frac{2}{65}$ | $\frac{1}{13}$ | $\frac{1}{65}$ | $\frac{1}{65}$ | $\frac{2}{65}$ | $\frac{3}{65}$ |
| 2 | $\frac{2}{65}$ | 0              | $\frac{1}{65}$ | 0              | 0              | 0              | $\frac{2}{65}$ |
| 3 | $\frac{1}{13}$ | $\frac{1}{65}$ | 0              | $\frac{2}{65}$ | 0              | 0              | $\frac{1}{13}$ |
| 4 | $\frac{1}{65}$ | 0              | $\frac{2}{65}$ | 0              | $\frac{4}{65}$ | 0              | 0              |
| 5 | $\frac{1}{65}$ | 0              | $\frac{3}{65}$ | $\frac{4}{65}$ | 0              |                |                |
| 6 | $\frac{2}{65}$ | 0              | 0              | 0              | 0              |                | $\frac{1}{65}$ |
| 7 | $\frac{3}{65}$ | $\frac{2}{65}$ | $\frac{1}{13}$ | 0              | 0              | $\frac{3}{65}$ | 0              |

$\mathcal{D}$

# Output: Constant-Degree DAN



# Objective: Expected Route Length

$$\text{ERL}(\mathcal{D}, N) = \sum_{(u,v) \in \mathcal{D}} p(u, v) \cdot d_N(u, v)$$

*DAN N* of degree  $\Delta$

path length on  $N$

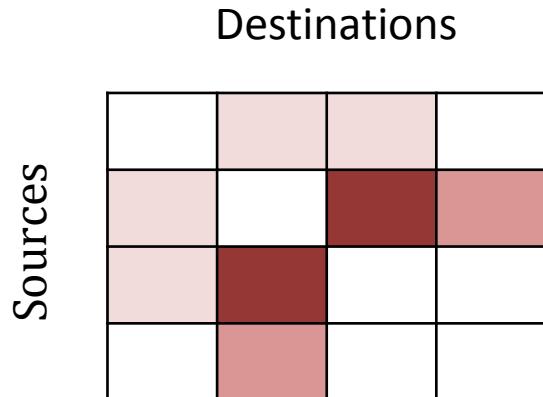
$\mathcal{D}[p(i, j)]$ : joint distribution

frequency

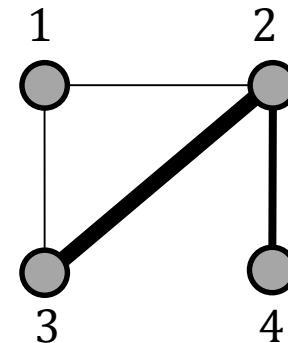
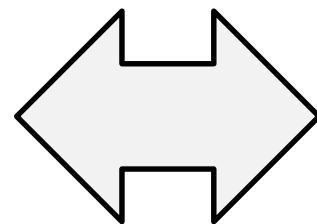
# Remark

- Can represent demand matrix as a **demand graph**

sparse distribution  $\mathcal{D}$

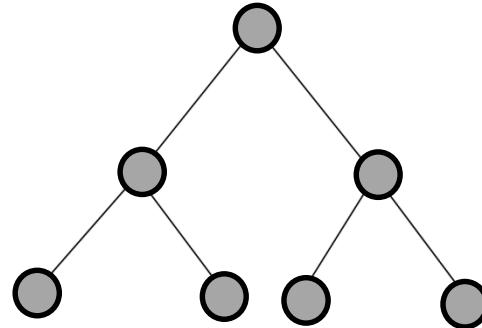


sparse graph  $G(\mathcal{D})$



# Some Examples

- DANs of  $\Delta = 3$ :
  - E.g., complete binary tree
  - $d_N(u,v) \leq 2 \log n$
  - Can we do **better** than ***log n***?
- DANs of  $\Delta = 2$ :
  - E.g., set of **lines** and **cycles**



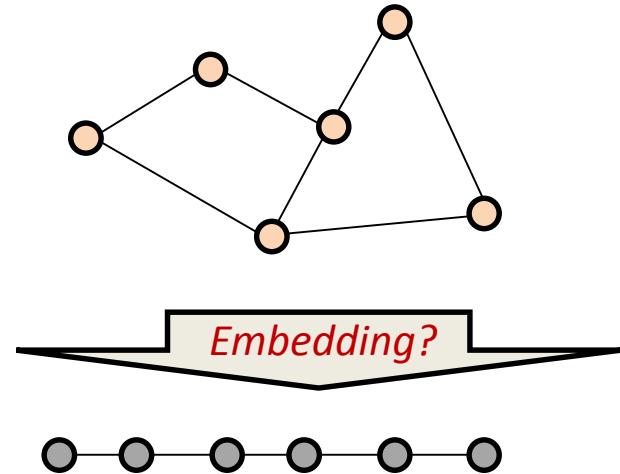
---

# Remark: Another Hardness Proof

---

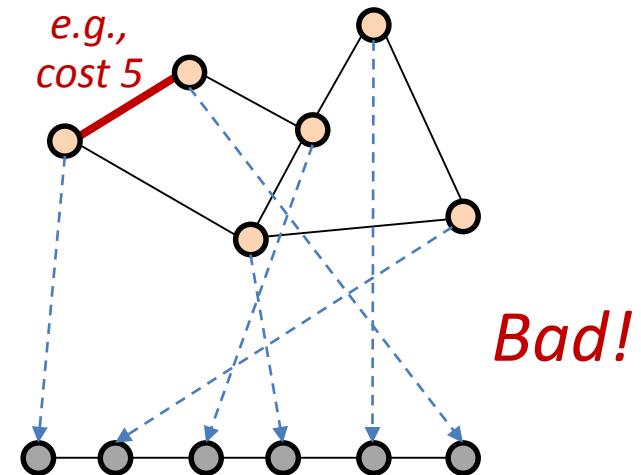
# DAN design can be NP-hard

- **Example  $\Delta = 2$ :** A Minimum Linear Arrangement (**MLA**) problem
  - A “Virtual Network Embedding Problem”, VNEP
  - *Minimize sum* of lengths of virtual edges



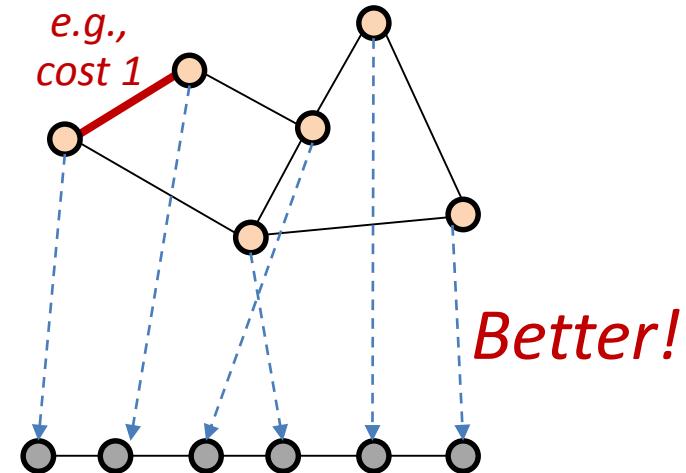
# DAN design can be NP-hard

- **Example  $\Delta = 2$ :** A Minimum Linear Arrangement (**MLA**) problem
  - A “Virtual Network Embedding Problem”, VNEP
  - *Minimize sum* of lengths of virtual edges



# DAN design can be NP-hard

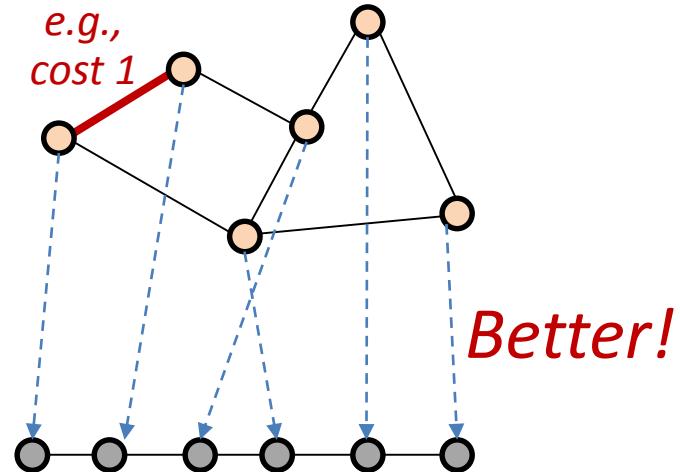
- **Example  $\Delta = 2$ :** A Minimum Linear Arrangement (**MLA**) problem
  - A “Virtual Network Embedding Problem”, VNEP
  - *Minimize sum* of lengths of virtual edges



# DAN design can be NP-hard

- Example  $\Delta = 2$ : A Minimum Linear Arrangement ( $MLA$ ) problem
  - A “Virtual Network Embedding Problem”, VNEP
  - *Minimizing the lengths of virtual edges*

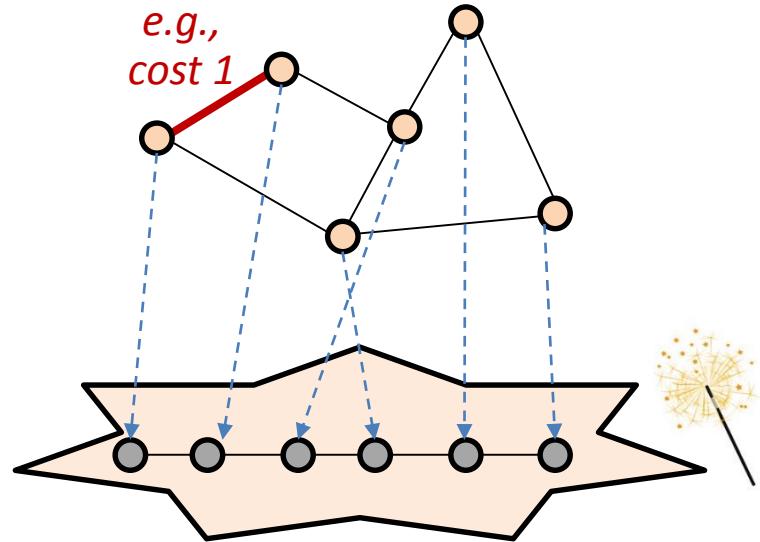
NP-hard, and so is DAN design.



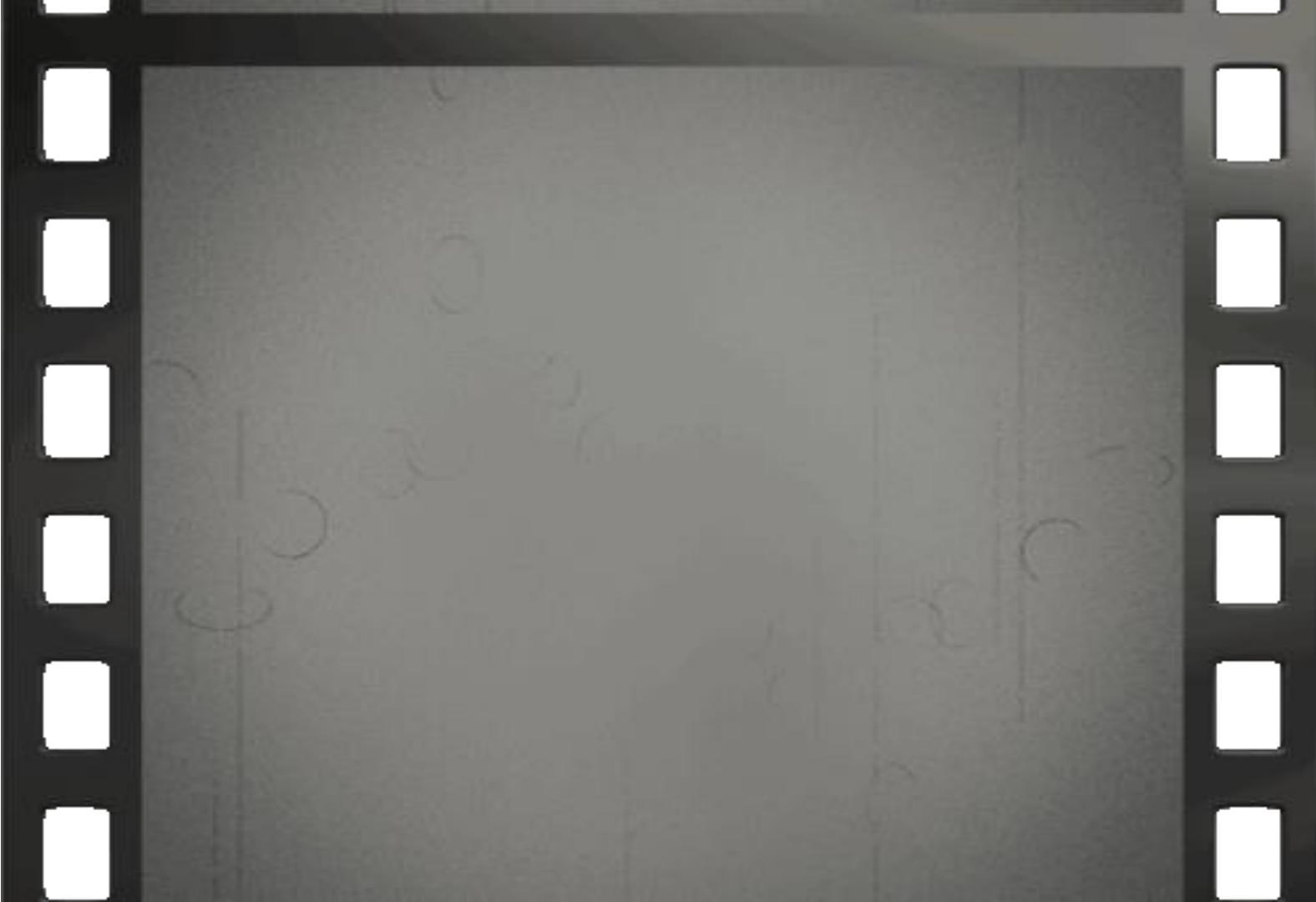
# DAN design can be NP-hard

- Example  $\Delta = 2$ : A Minimum Linear Arrangement (MLA) problem
  - A “Virtual Network Embedding Problem”, VNEP
  - *Minimizing lengths of virtual edges*
- But what about  $> 2$ ? *Embedding* problem still hard, but we have an additional **degree of freedom**:

Do topological flexibilities make problem easier or harder?!



*A new knob for optimization!*

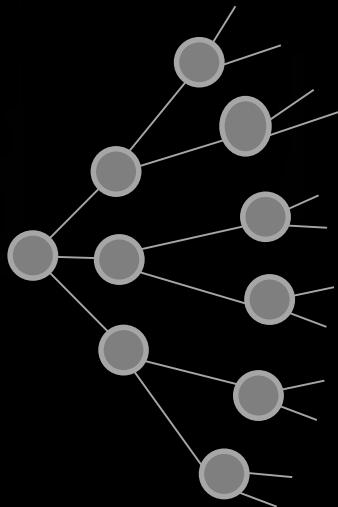


# Rewinding the Clock: Degree-Diameter Tradeoff

Each network with  $n$  nodes and max degree  $\Delta > 2$  must have a diameter of at least  $\log(n)/\log(\Delta-1)-1$ .

Example: constant  $\Delta$ ,  $\log(n)$  diameter

# Proof Idea



$$1 \quad \Delta \quad \Delta(\Delta - 1) \dots$$

In  $k$  steps,  
reach at most  
 $1 + \sum \Delta(\Delta - 1)^k$   
nodes

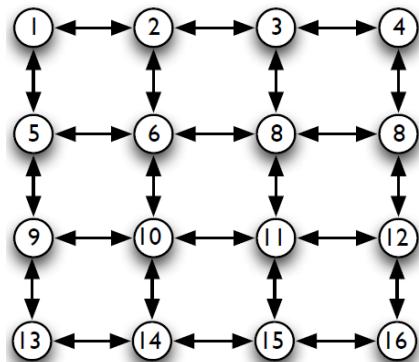
---

Is there a better tradeoff in DANs?

---

# Sometimes, DANs can be much better!

## Example 1: low-degree demand

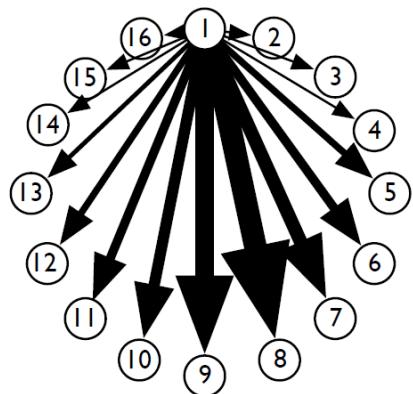


If **demand graph** is of degree  $\Delta$ , it is trivial to design a **DAN** of degree  $\Delta$  which achieves an *expected route length of 1*.

Just take DAN =  
demand graph!

# Sometimes, DANs can be much better!

## Example 2: skewed demand

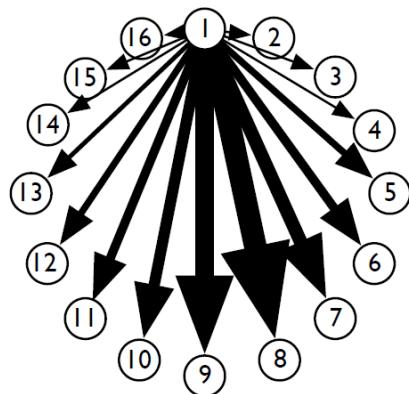


If **demand** is highly skewed, it is also possible to achieve an ***expected route length of 1*** in a constant-degree DAN.



# Sometimes, DANs can be much better!

## Example 2: skewed demand



If **demand** is highly skewed, it is also possible to achieve an ***expected route length of 1*** in a constant-degree DAN.



E.g., arrange neighbors of node 1 in a **Huffman tree**!

---

So on what does it depend?

---

# So on what does it depend?



We argue (but still don't know!): on the  
**“entropy” of the demand!**



SPEED  
LIMIT  
?

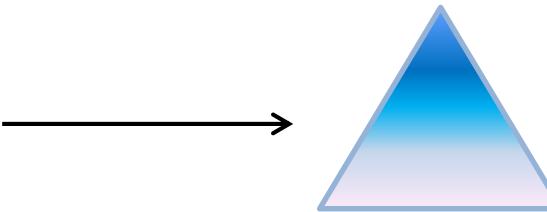
# Intuition: Entropy Lower Bound



# Lower Bound Idea: Leverage Coding or Datastructure

|         |   | Destinations   |                |                |                |                |                |                |
|---------|---|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
|         |   | 1              | 2              | 3              | 4              | 5              | 6              | 7              |
| Sources | 1 | 0              | $\frac{2}{65}$ | $\frac{1}{13}$ | $\frac{1}{65}$ | $\frac{1}{65}$ | $\frac{2}{65}$ | $\frac{3}{65}$ |
|         | 2 | $\frac{2}{65}$ | 0              | $\frac{1}{65}$ | 0              | 0              | 0              | $\frac{2}{65}$ |
|         | 3 | $\frac{1}{13}$ | $\frac{1}{65}$ | 0              | $\frac{2}{65}$ | 0              | 0              | $\frac{1}{13}$ |
|         | 4 | $\frac{1}{65}$ | 0              | $\frac{2}{65}$ | 0              | $\frac{4}{65}$ | 0              | 0              |
|         | 5 | $\frac{1}{65}$ | 0              | $\frac{3}{65}$ | $\frac{4}{65}$ | 0              | 0              | 0              |
|         | 6 | $\frac{2}{65}$ | 0              | 0              | 0              | 0              | 0              | $\frac{3}{65}$ |
|         | 7 | $\frac{3}{65}$ | $\frac{2}{65}$ | $\frac{1}{13}$ | 0              | 0              | $\frac{3}{65}$ | 0              |

- DAN just for a *single (source) node 3*



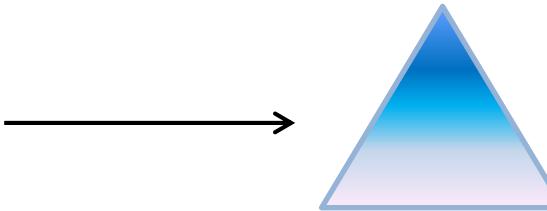
- How good can this tree be? Cannot do better than  $\Delta$ -ary **Huffman tree** for its destinations
- Entropy** lower bound on ERL known for binary trees, e.g. **Mehlhorn** 1975

# Lower Bound Idea: Leverage Coding or Datastructure

An optimal “ego-tree”  
for this source!

|         |   | Destinations   |                |                |                |                |                |                |
|---------|---|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
|         |   | 1              | 2              | 3              | 4              | 5              | 6              | 7              |
| Sources | 1 | 0              | $\frac{2}{65}$ | $\frac{1}{13}$ | $\frac{1}{65}$ | $\frac{1}{65}$ | $\frac{2}{65}$ | $\frac{3}{65}$ |
|         | 2 | $\frac{2}{65}$ | 0              | $\frac{1}{65}$ | 0              | 0              | 0              | $\frac{2}{65}$ |
|         | 3 | $\frac{1}{13}$ | $\frac{1}{65}$ | 0              | $\frac{2}{65}$ | 0              | 0              | $\frac{1}{13}$ |
|         | 4 | $\frac{1}{65}$ | 0              | $\frac{2}{65}$ | 0              | $\frac{4}{65}$ | 0              | 0              |
|         | 5 | $\frac{1}{65}$ | 0              | $\frac{3}{65}$ | $\frac{4}{65}$ | 0              | 0              | 0              |
|         | 6 | $\frac{2}{65}$ | 0              | 0              | 0              | 0              | 0              | $\frac{3}{65}$ |
|         | 7 | $\frac{3}{65}$ | $\frac{2}{65}$ | $\frac{1}{13}$ | 0              | 0              | $\frac{3}{65}$ | 0              |

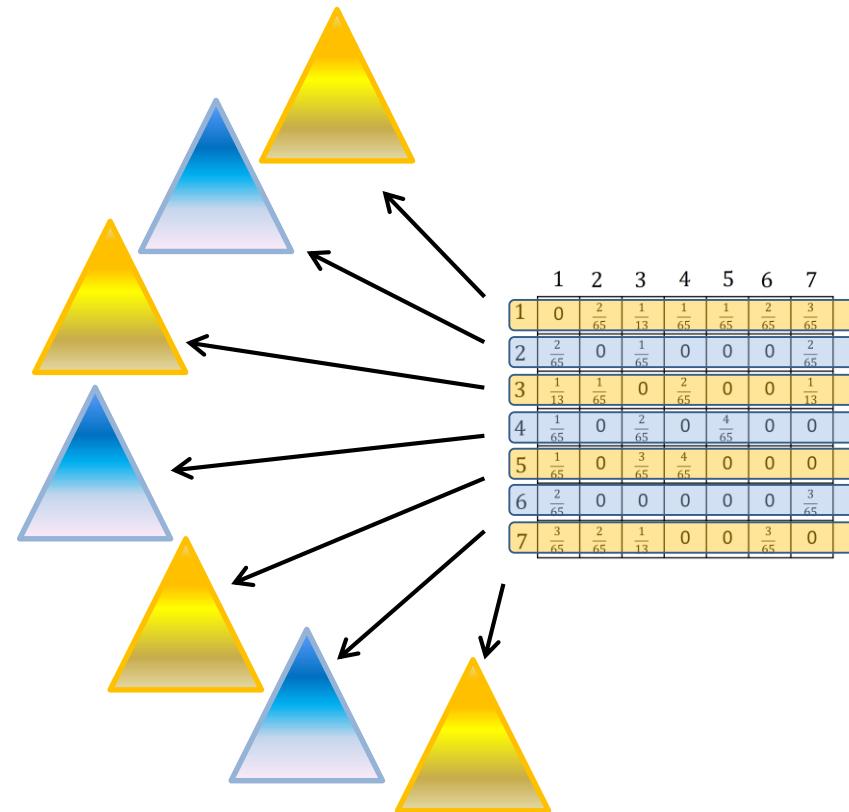
- DAN just for a *single (source) node 3*



- How good can this tree be? Cannot do better than  $\Delta$ -ary **Huffman tree** for its destinations
- Entropy** lower bound on ERL known for binary trees, e.g. **Mehlhorn** 1975

# So: Entropy of the *Entire* Demand

- Proof idea ( $EPL = \Omega(H_{\Delta}(Y|X))$ ):
  - sources
  - destinations
  - entropy
  - degree
- Compute **ego-tree** for each source node
- Take **union** of all **ego-trees**
- Violates **degree restriction** but valid lower bound



# Entropy of the *Entire* Demand: Sources and Destinations

Do this in **both dimensions**:  
 $EPL \geq \Omega(\max\{H_{\Delta}(Y|X), H_{\Delta}(X|Y)\})$

|   |   | $\Omega(H_{\Delta}(X Y))$ |                |                |                |                |                |                |  |
|---|---|---------------------------|----------------|----------------|----------------|----------------|----------------|----------------|--|
|   |   | 1                         | 2              | 3              | 4              | 5              | 6              | 7              |  |
| 1 | 1 | 0                         | $\frac{2}{65}$ | $\frac{1}{13}$ | $\frac{1}{65}$ | $\frac{1}{65}$ | $\frac{2}{65}$ | $\frac{3}{65}$ |  |
|   | 2 | $\frac{2}{65}$            | 0              | $\frac{1}{65}$ | 0              | 0              | 0              | $\frac{2}{65}$ |  |
| 3 | 3 | $\frac{1}{13}$            | $\frac{1}{65}$ | 0              | $\frac{2}{65}$ | 0              | 0              | $\frac{1}{13}$ |  |
|   | 4 | $\frac{1}{65}$            | 0              | $\frac{2}{65}$ | 0              | $\frac{4}{65}$ | 0              | 0              |  |
| 5 | 5 | $\frac{1}{65}$            | 0              | $\frac{3}{65}$ | $\frac{4}{65}$ | 0              | 0              | 0              |  |
|   | 6 | $\frac{2}{65}$            | 0              | 0              | 0              | 0              | 0              | $\frac{3}{65}$ |  |
| 7 | 7 | $\frac{3}{65}$            | $\frac{2}{65}$ | $\frac{1}{13}$ | 0              | 0              | $\frac{3}{65}$ | 0              |  |

$\mathcal{D}$

$\Omega(H_{\Delta}(Y|X))$

# Entropy of the *Entire* Demand: Sources and Destinations

Do this in **both dimensions**:  
 $EPL \geq \Omega(\max\{H_{\Delta}(Y|X), H_{\Delta}(X|Y)\})$

|   |   | $\Omega(H_{\Delta}(X Y))$ |                |                |                |                |                |                |  |
|---|---|---------------------------|----------------|----------------|----------------|----------------|----------------|----------------|--|
|   |   | 1                         | 2              | 3              | 4              | 5              | 6              | 7              |  |
| 1 | 1 | 0                         | $\frac{2}{65}$ | $\frac{1}{13}$ | $\frac{1}{65}$ | $\frac{1}{65}$ | $\frac{2}{65}$ | $\frac{3}{65}$ |  |
|   | 2 | $\frac{2}{65}$            | 0              | $\frac{1}{65}$ | 0              | 0              | 0              | $\frac{2}{65}$ |  |
| 3 | 3 | $\frac{1}{13}$            | $\frac{1}{65}$ | 0              | $\frac{2}{65}$ | 0              | 0              | $\frac{1}{13}$ |  |
|   | 4 | $\frac{1}{65}$            | 0              | $\frac{2}{65}$ | 0              | $\frac{4}{65}$ | 0              | 0              |  |
| 5 | 5 | $\frac{1}{65}$            | 0              | $\frac{3}{65}$ | $\frac{4}{65}$ | 0              | 0              | 0              |  |
|   | 6 | $\frac{2}{65}$            | 0              | 0              | 0              | 0              | 0              | $\frac{3}{65}$ |  |
| 7 | 7 | $\frac{3}{65}$            | $\frac{2}{65}$ | $\frac{1}{13}$ | 0              | 0              | $\frac{3}{65}$ | 0              |  |

$\mathcal{D}$

$\Omega(H_{\Delta}(Y|X))$

---

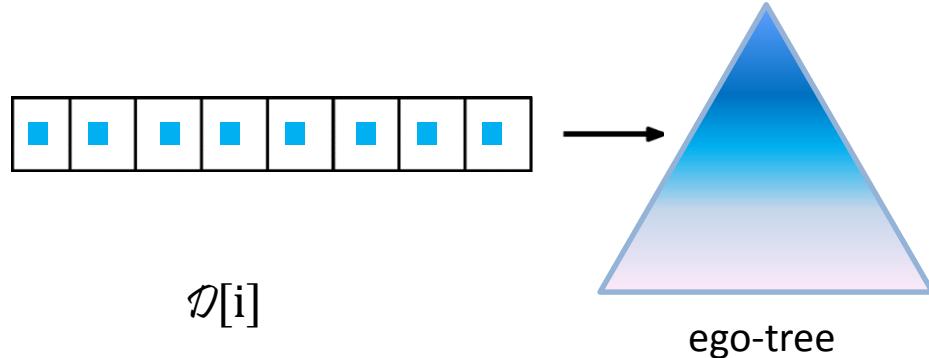
# Achieving Entropy Limit: Algorithms

---



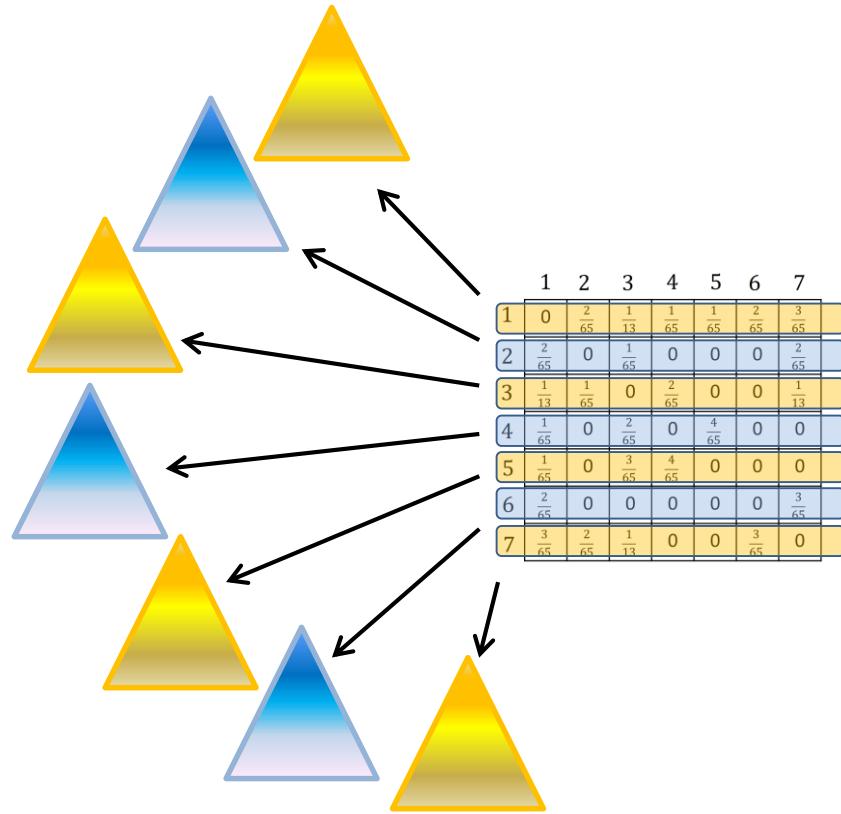
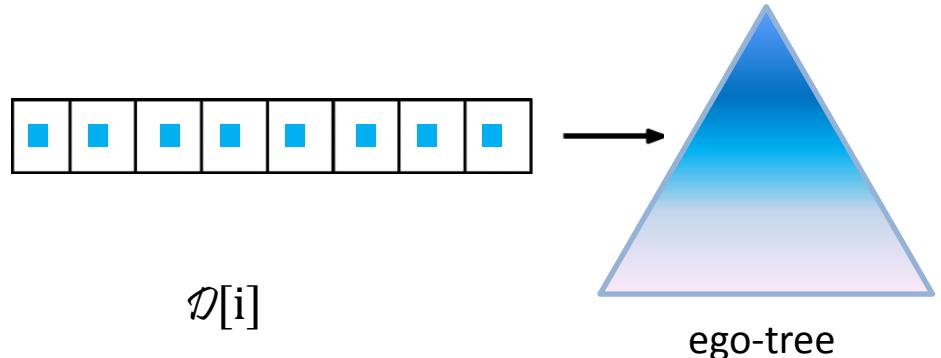
# Ego-Trees Revisited

- ego-tree: optimal tree for a row (= given source)



# Ego-Trees Revisited

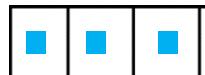
- ego-tree: optimal tree for a row (= given source)



Can we merge the trees **without distortion** and **keep degree low**?

# Ego-Trees Revisited

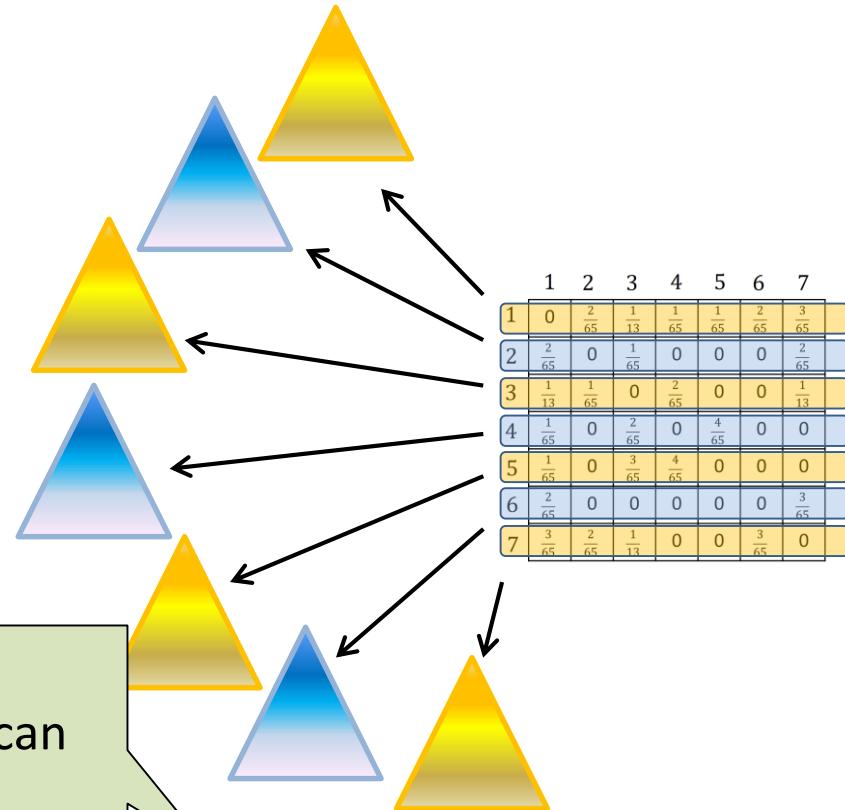
- ego-tree: optimal tree for a row (= given source)



$\mathcal{D}[i]$

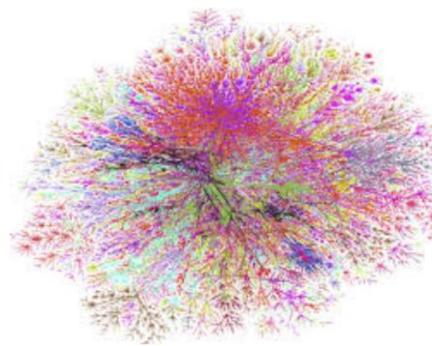
For **sparse demands** yes:  
enough **low-degree nodes** which can  
serve as “**helper nodes**”!

ego-tree



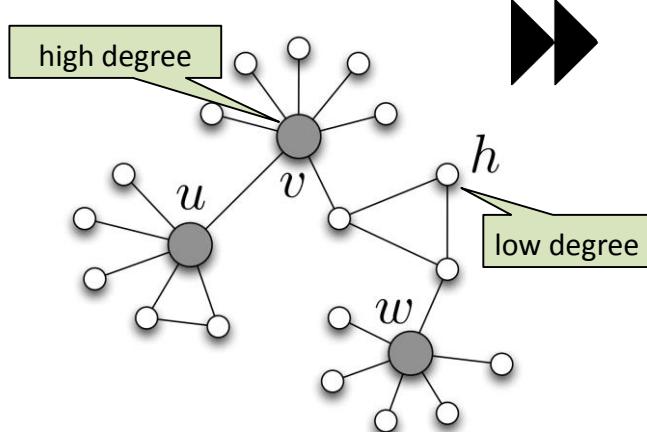
Can we merge the trees **without distortion** and **keep degree low**?

# From Trees to Networks

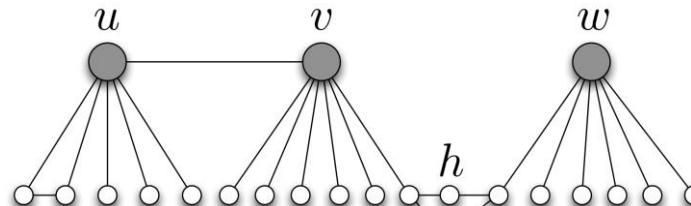


# Idea: Degree Reduction

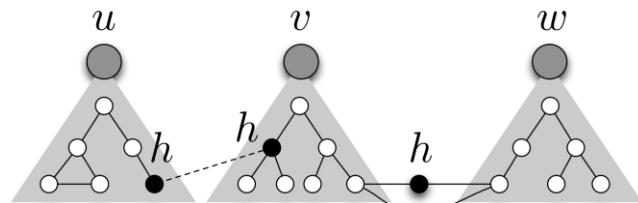
① Demand graph



② Hierarchical representation



③ Add low-degree nodes as helpers

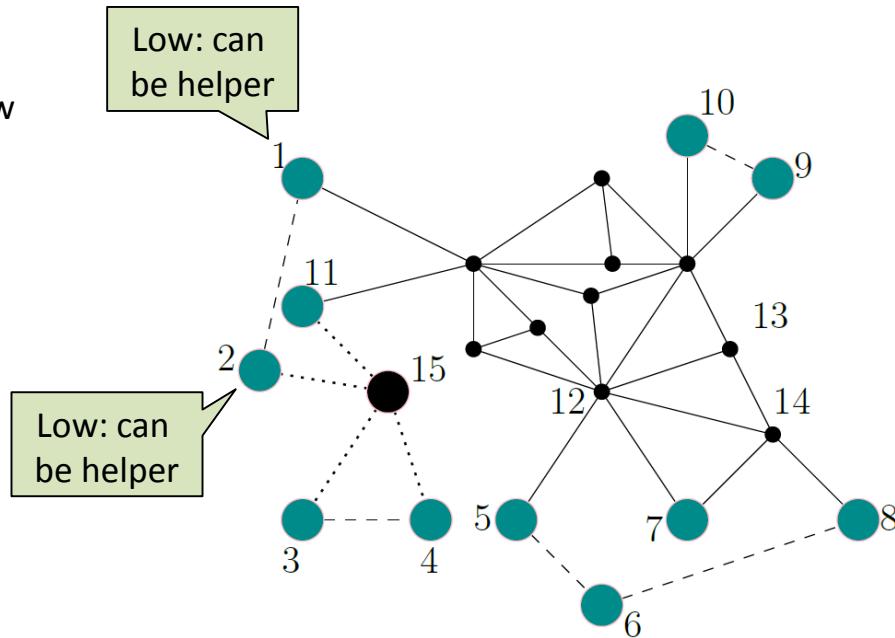


Taking union of ego-trees results in **high degree**:  
 $u$  and  $v$  will appear in many ego-trees

Node  $h$  **helps edge  $(u, v)$**  by participating in **ego-tree( $u$ )** as a relay node toward  $v$  and **in ego-tree( $v$ )** as a relay toward  $u$

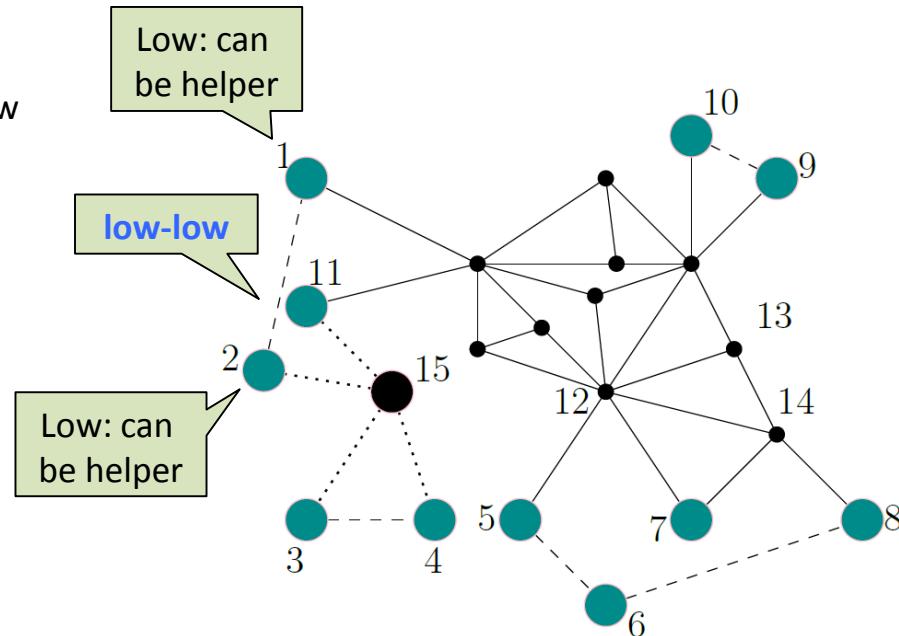
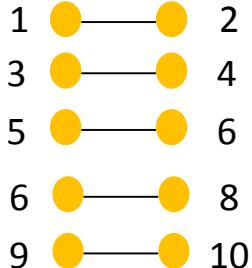
# Algorithm: Degree Reduction

- Find **low** degree nodes
  - Half of the nodes of lowest degree: “below twice average degree”



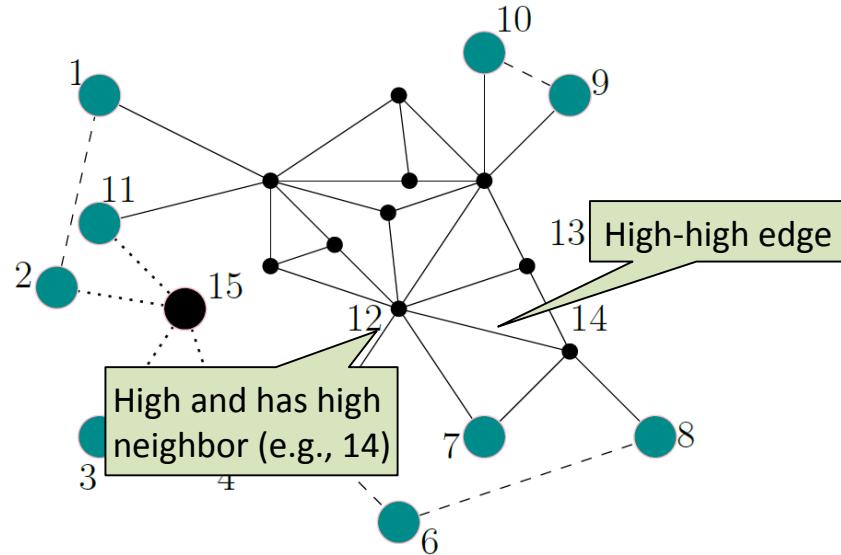
# Algorithm: Degree Reduction

- Find **low** degree nodes
    - Half of the nodes of lowest degree: “below twice average degree”
  - Put the **low-low** edges into DAN and remove from demand



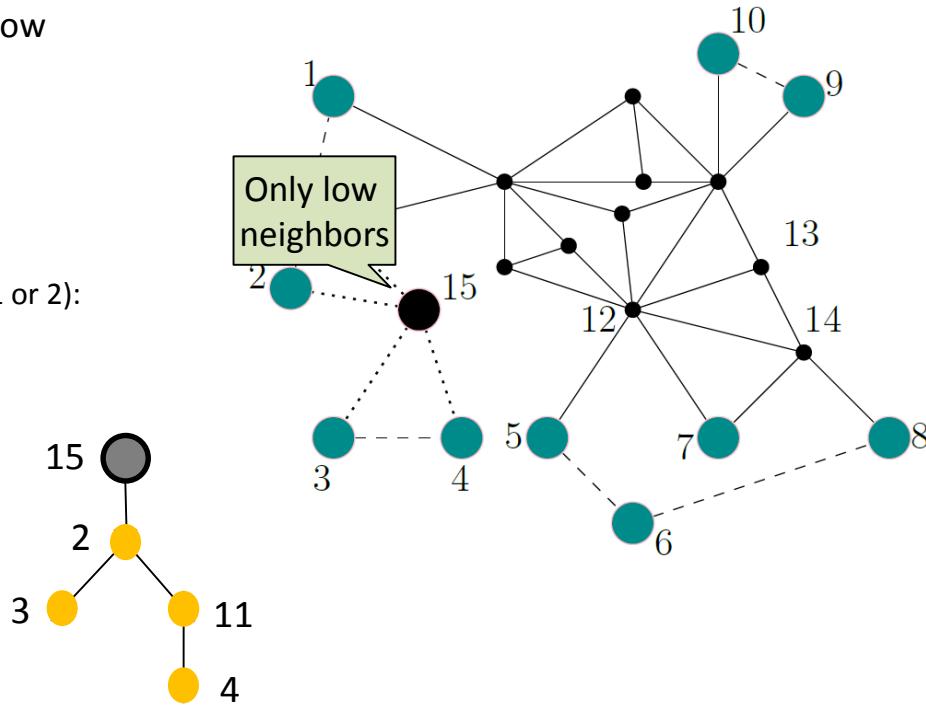
# Algorithm: Degree Reduction

- Find **low** degree nodes
  - Half of the nodes of lowest degree: “below twice average degree”
- **Put** the **low-low** edges into DAN and remove from demand
- Mark **high-high** edges
  - Put (any) **low degree** nodes in between (e.g., 1 or 2): one is enough so distance increased by **+1**



# Algorithm: Degree Reduction

- Find **low** degree nodes
  - Half of the nodes of lowest degree: “below twice average degree”
- **Put** the **low-low** edges into DAN and remove from demand
- Mark **high-high** edges
  - Put (any) **low degree** nodes in between (e.g., 1 or 2): one is enough so distance increased by **+1**
- Now high degree nodes have only low degree neighbors: make **tree**
  - Create optimal **binary tree** with low degree neighbors



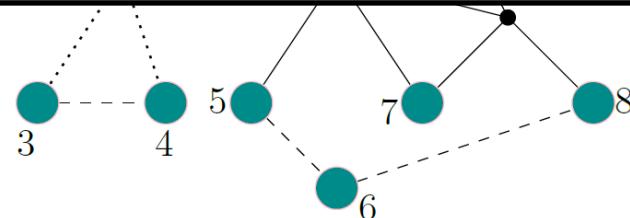
# Algorithm: Degree Reduction

- Find **low** degree nodes
  - Half of the nodes of lowest degree: “below twice average degree”



**Theorem [Asymptotic Optimality]:** Helper node does not participate in many trees, so ***constant degree***, and ***constant distortion***.

- Now high degree nodes have only low degree neighbors: make **tree**
  - Create optimal **binary tree** with low degree neighbors



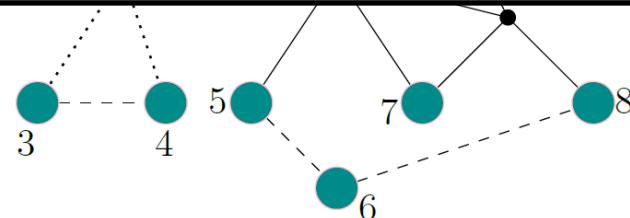
# Algorithm: Degree Reduction

- Find **low** degree nodes
  - Half of the nodes of lowest degree: “below twice average degree”



**Theorem [Asymptotic Optimality]:** Helper node does not participate in many trees, so ***constant degree***, and ***constant distortion***.

- Now high degree nodes have only low degree neighbors: make **tree**
  - Create optimal **binary tree** with low degree neighbors





---

# DAN Design: Related to Spanners

---

# Low-Distortion Spanners

- Classic problem: find sparse, distance-preserving (low-distortion) **spanner** of a graph

The „DAN“

The demand  
graph

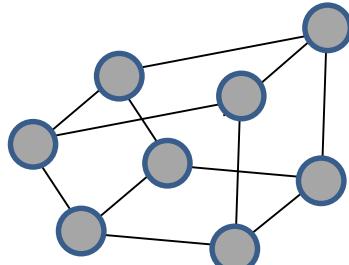
# Low-Distortion Spanners

- Classic problem: find sparse, distance-preserving (low-distortion) **spanner** of a graph
- But:
  - Spanners aim at low distortion among ***all pairs***; in our case, we are only interested in the ***local distortion***, 1-hop communication neighbors
  - We allow **auxiliary edges** (not a subgraph): similar to **geometric spanners**
  - We require ***constant degree***

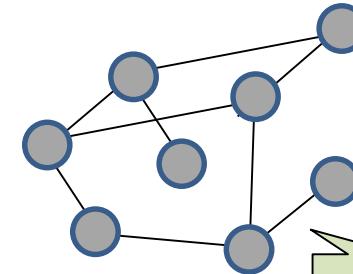
# Yet: We can leverage the connection to spanners sometimes!

**Theorem:** If request distribution  $\mathcal{D}$  is **regular and uniform**, and if we can find a constant distortion, linear sized (i.e., **constant, sparse**) spanner for this request graph: then we can design a constant degree DAN providing an **optimal ERL** (i.e.,  $O(H(X|Y)+H(Y|X))$ ).

*r-regular and uniform*  
demand:

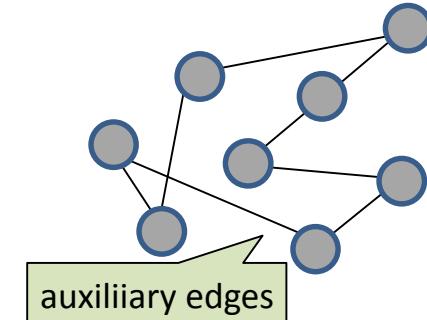


*Sparse, irregular  
(constant) spanner:*



subgraph!

*Constant degree optimal*  
DAN (ERL at most **log r**):



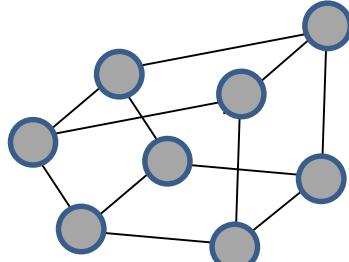
auxiliary edges

# Yet: We can leverage the connection to spanners sometimes!

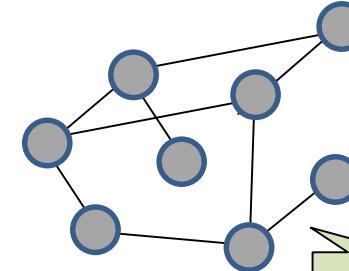
**Theorem:** If request distribution  $\mathcal{D}$  is **regular and uniform**, and if we can find a constant distortion, linear sized (i.e., **constant sparse**) spanner for this request graph: then we can design a constant degree DAN

Simply using our degree reduction trick again: now for spanner!

***r-regular and uniform***  
demand:

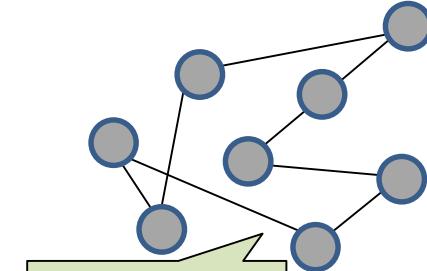


***Sparse, irregular  
(constant) spanner:***



subgraph!

***Constant degree optimal  
DAN (ERL at most log r):***

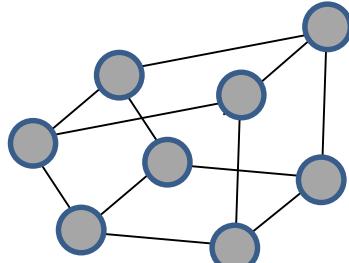


auxiliary edges

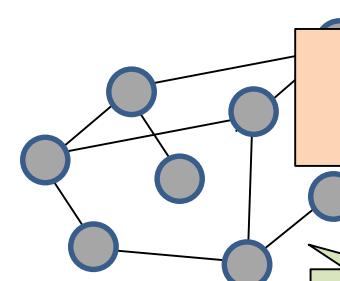
# Yet: We can leverage the connection to spanners sometimes!

**Theorem:** If request distribution  $\mathcal{D}$  is **regular and uniform**, and if we can find a constant distortion, linear sized (i.e., **constant, sparse**) spanner for this request graph: then we can design a constant degree DAN providing an **optimal EPL** (i.e.,  $O(H(X|Y)+H(Y|X))$ ).

*r-regular and uniform*  
demand:



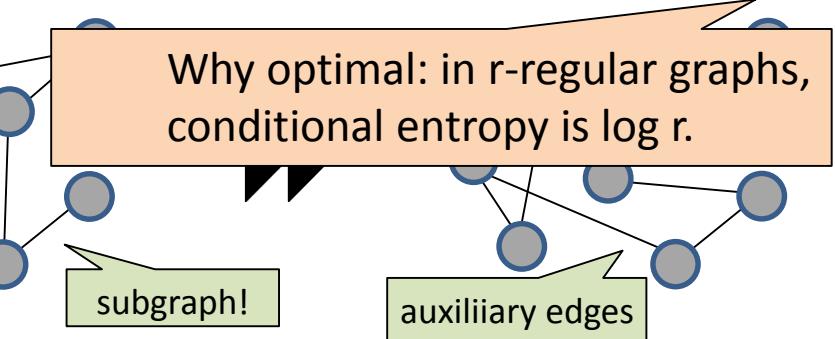
*Sparse, irregular  
(constant) spanner:*



Why optimal: in  $r$ -regular graphs,  
conditional entropy is  $\log r$ .

subgraph!

*Constant degree optimal*  
DAN (ERL at most  **$\log r$** ):

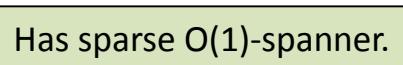


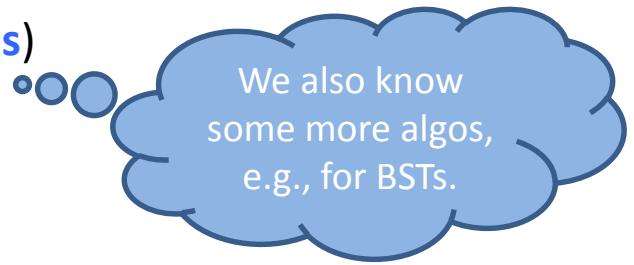
auxiliary edges

# Proof Idea

- **Degree reduction** again, this time *from sparse spanner* (before: from sparse demand graph)

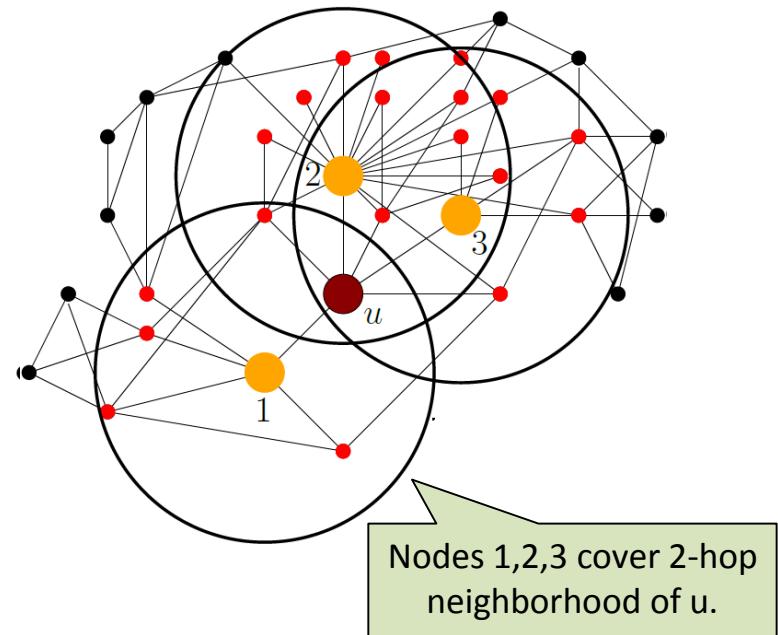
## Corollaries

- Optimal DAN designs for
  - **Hypercubes** (with  $n \log n$  edges)
  - **Chordal graphs**  Has sparse  $O(1)$ -spanner.
  - Trivial: graphs with polynomial degree (**dense graphs**)
  - Graphs of **locally bounded doubling dimension**



# An Example: Demands of Locally-Bounded Doubling Dimension

- LDD:  $G_{\emptyset}$  has a **Locally-bounded Doubling Dimension** (LDD) iff all 2-hop neighbors are covered by 1-hop neighbors of just  $\lambda$  nodes
  - Note: care only about **2-neighborhood**
- Formally,  $B(u, 2) \subseteq \bigcup_{i=1}^{\lambda} B(v_i, 1)$
- Challenge: can be of **high degree**!



# DAN for Locally-Bounded Doubling Dimension

**Lemma:** There exists a sparse 9-(subgraph)spanner for LDD.

This *implies optimal DAN*: still focus on regular and uniform!

**Def. ( $\varepsilon$ -net):** A subset  $V'$  of  $V$  is a  **$\varepsilon$ -net** for a graph  $G = (V, E)$  if

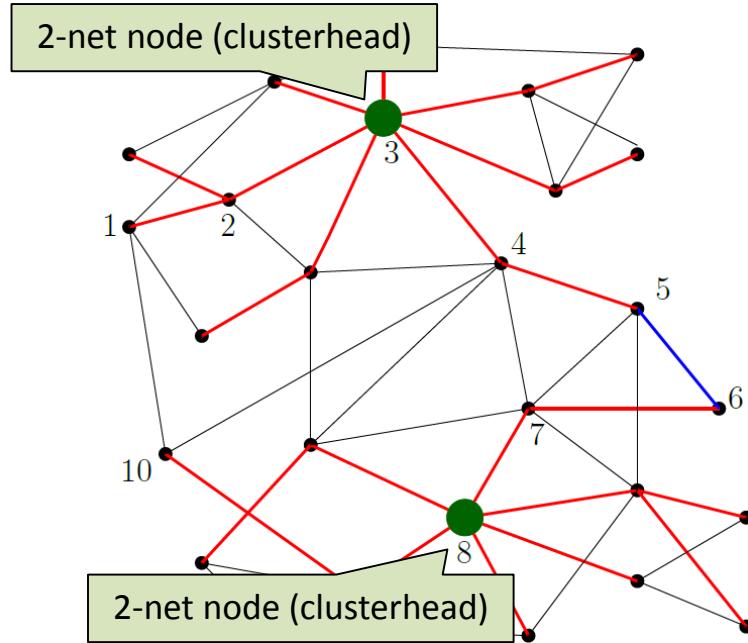
- $V'$  sufficiently “**independent**”: for every  $u, v \in V'$ ,  $d_G(u, v) > \varepsilon$
- “**dominating**”  $V$ : for each  $w \in V$ ,  $\exists$  at least one  $u \in V'$  such that,  $d_G(u, w) \leq \varepsilon$

# 9-Spanner for LDD (= optimal DAN)

## Simple algorithm:

### 1. Find a 2-net

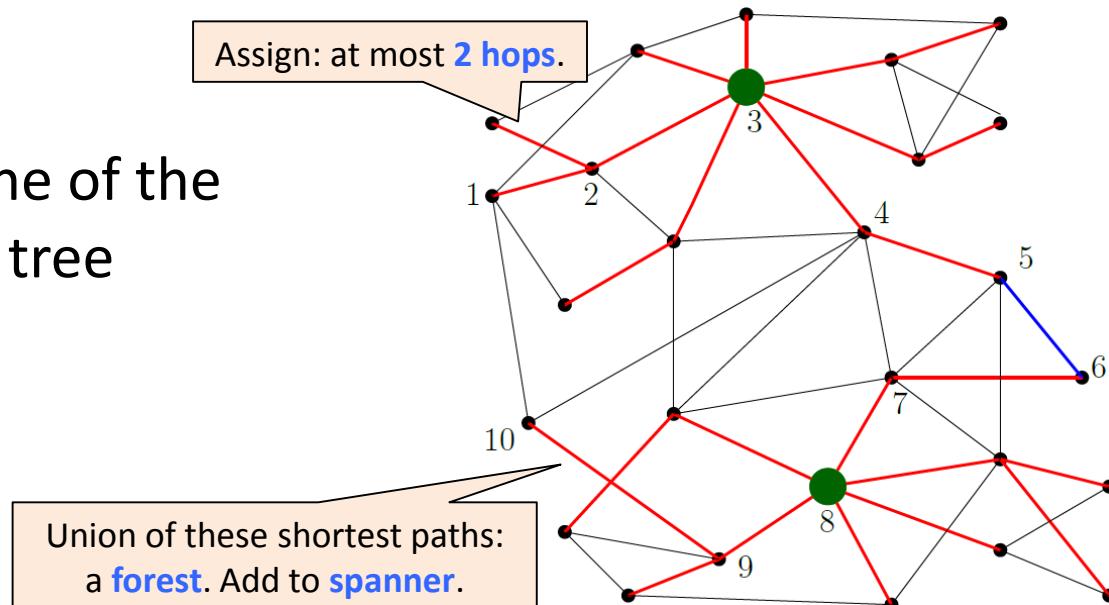
Easy: Select nodes into 2-net **one-by-one** in decreasing (remaining) degrees, **remove 2-neighborhood**. Iterate.



# 9-Spanner for LDD (= optimal DAN)

## Simple algorithm:

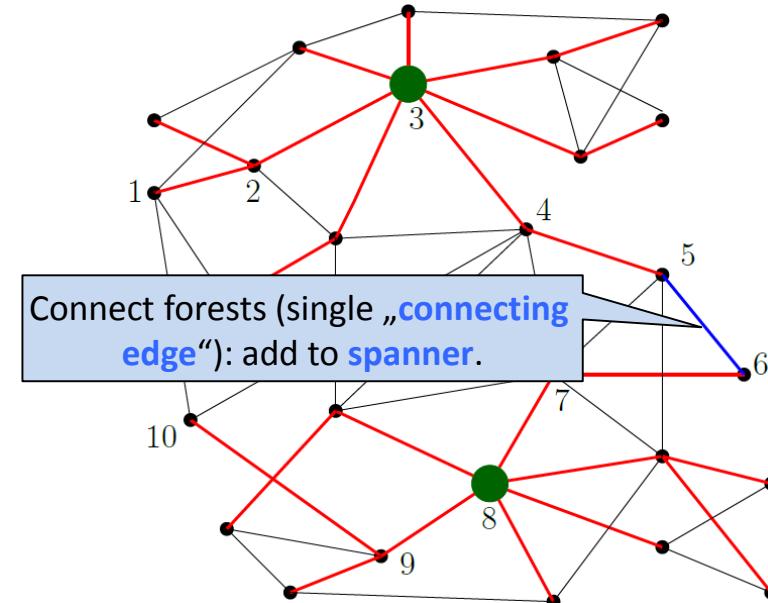
1. Find a 2-net
2. Assign nodes to one of the closest 2-net nodes: tree



# 9-Spanner for LDD (= optimal DAN)

**Simple algorithm:**

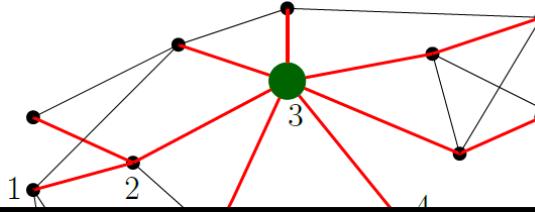
1. Find a 2-net
2. Assign nodes to one of the closest 2-net nodes: tree
3. Join two clusters if there are edges in between



# 9-Spanner for LDD (= optimal DAN)



Distortion 9: *Short detour* via  
clusterheads:  $u, ch(u), x, y, ch(v), v$



1. Build a 2-net

2. Assign nodes to one of the closest 2-net nodes

3. Join two clusters  
edges in between



**Sparse:** Spanner only includes *forest* (sparse) plus “connecting edges”: but since in *a locally doubling dimension graph* the number of cluster heads at distance 5 is bounded, only a small number of neighboring clusters will communicate.



# So: How *much* structure/entropy is there?



How to *measure* it?  
And which *types of structures*? E.g., **temporal**  
structure in addition to **non-temporal** structure?  
More *tricky*!

# Often only intuitions in the literature...

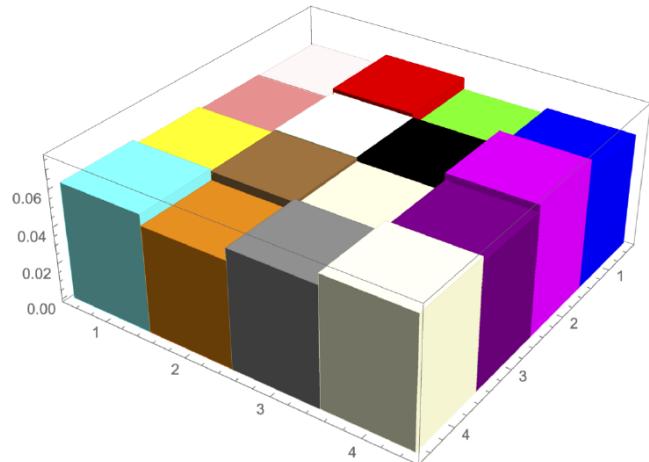
*“less than 1% of the rack pairs account for 80% of the total traffic”*

*“only a few ToRs switches are hot and most of their traffic goes to a few other ToRs”*

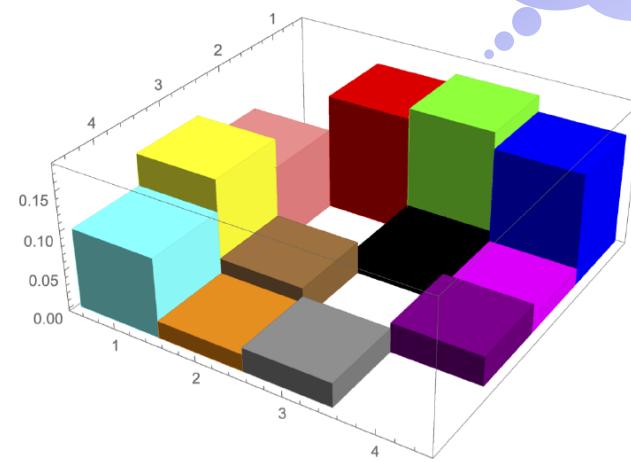
*“over 90% bytes flow in elephant flows”*

# ... and it *is* intuitive!

## Non-temporal Structure



VS



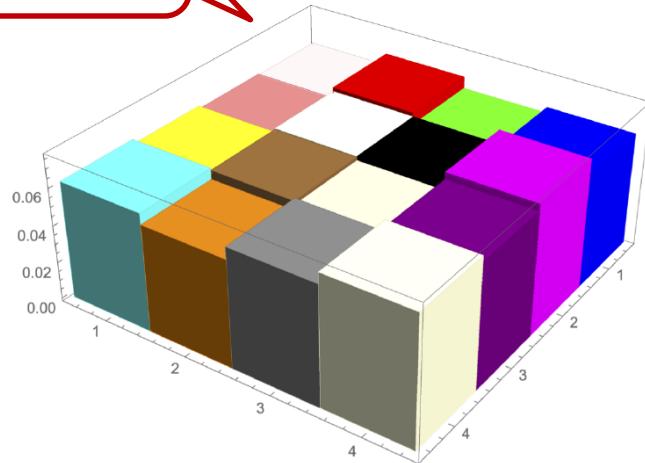
Color =  
comm. pair

Traffic matrix of two different **distributed ML** applications (GPU-to-GPU):  
Which one has *more structure*?

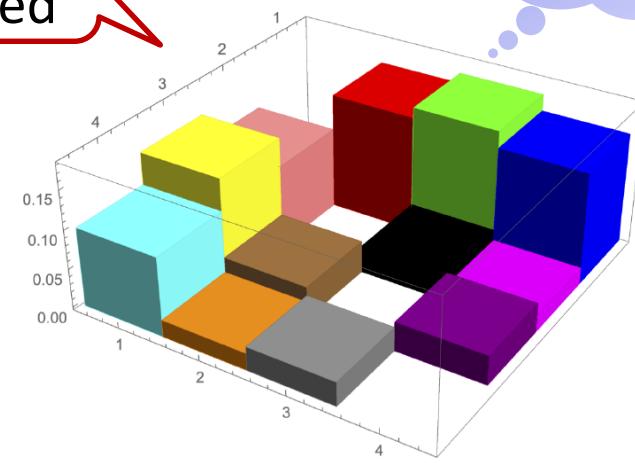
# ... and it *is* intuitive!

## Non-temporal Structure

More uniform



More skewed



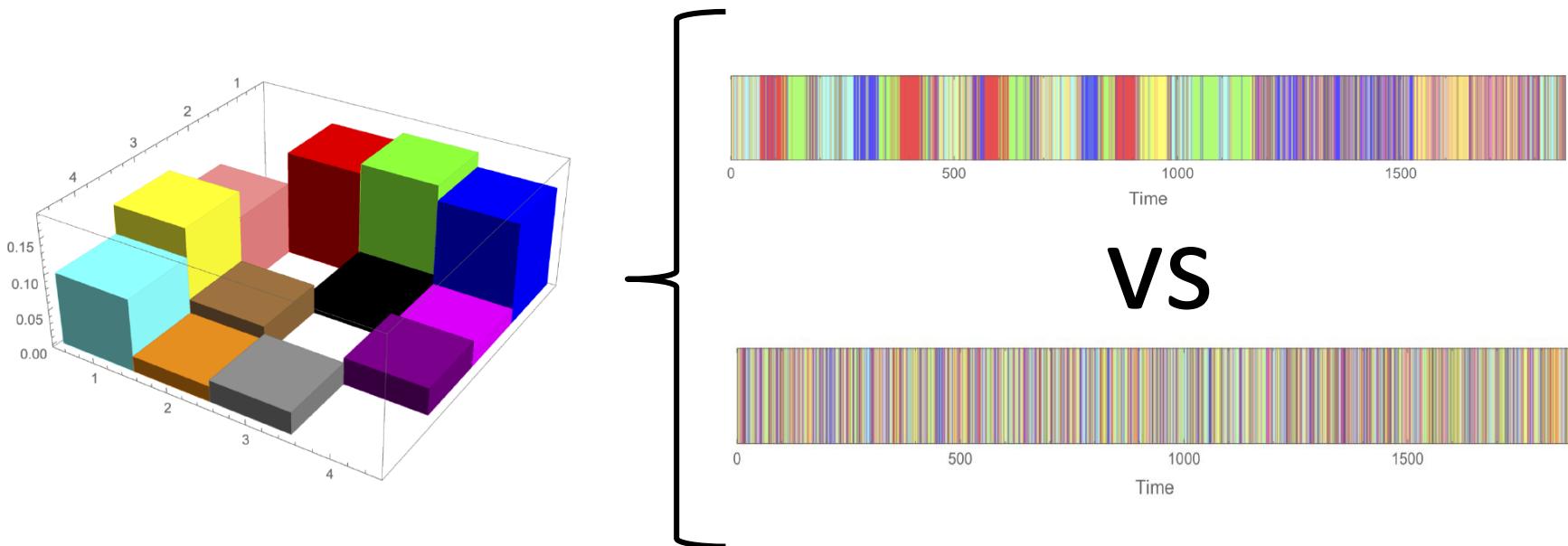
VS

Color =  
comm. pair

Traffic matrix of two different **distributed ML** applications (GPU-to-GPU):  
Which one has *more structure*?

# ... and it *is* intuitive!

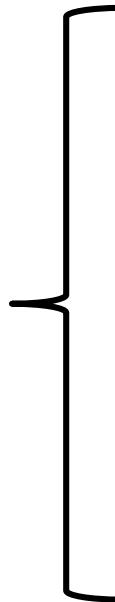
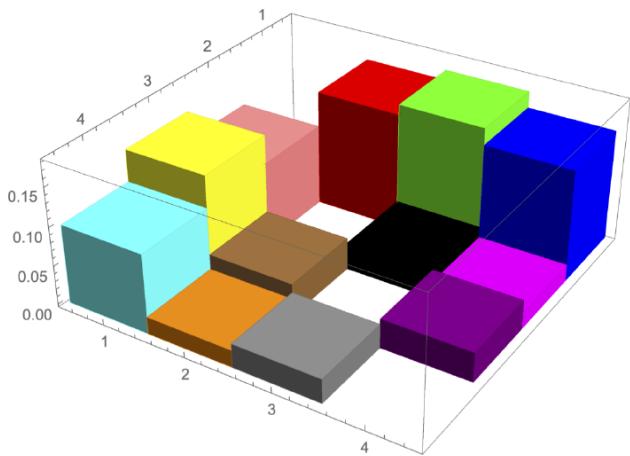
## Temporal Structure



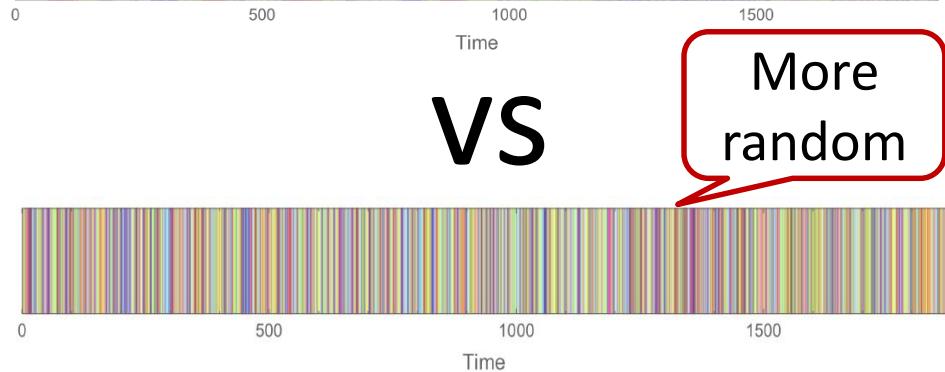
Two different ways to generate *same traffic matrix* (same non-temporal structure):  
Which one has *more structure*?

# ... and it *is* intuitive!

## Temporal Structure



VS



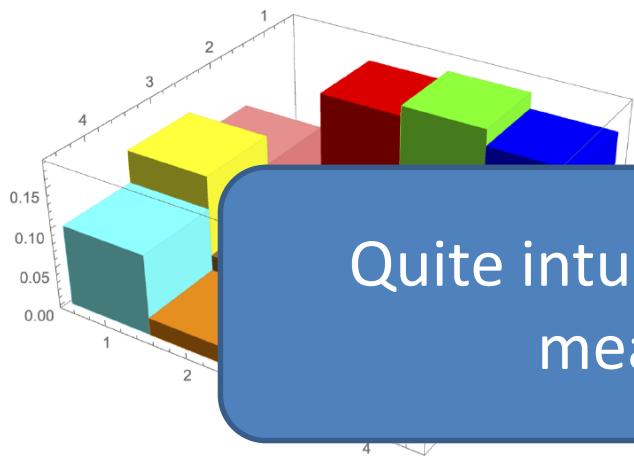
More  
bursty

More  
random

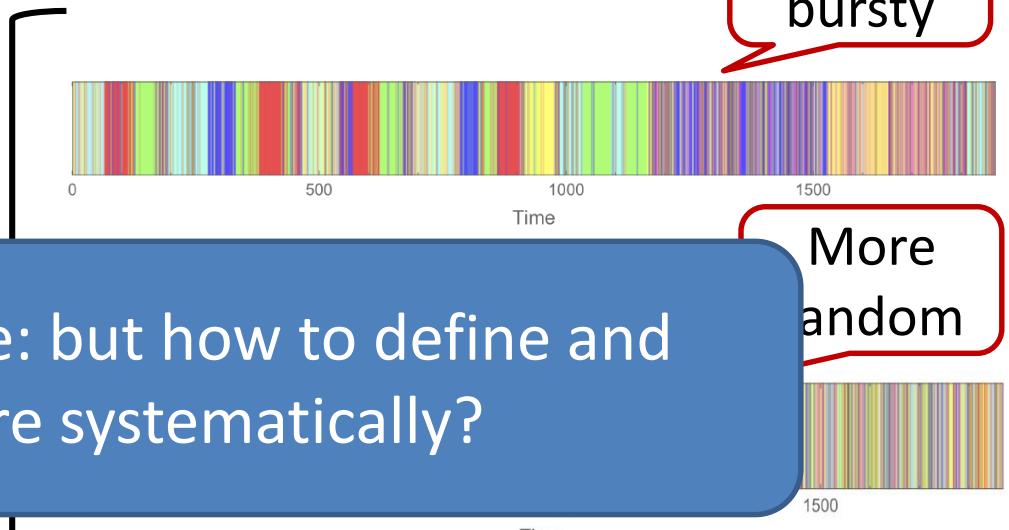
Two different ways to generate *same traffic matrix* (same non-temporal structure):  
Which one has *more structure*?

# ... and it *is* intuitive!

## Temporal Structure



Quite intuitive: but how to define and measure systematically?



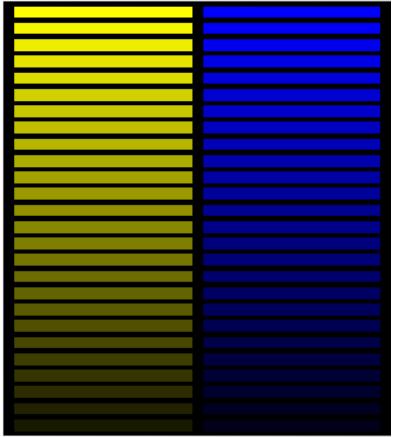
Two different ways to generate *same traffic matrix* (same non-temporal structure):  
Which one has *more structure*?

# The Trace Complexity

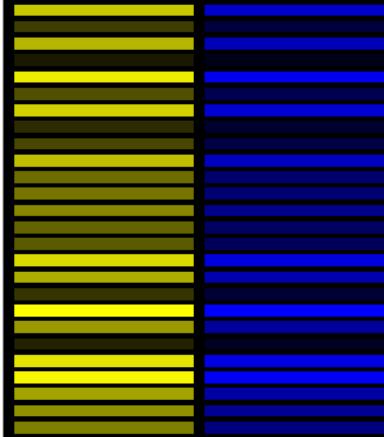
- An **information-theoretic** approach: how can we ***measure the entropy*** (rate) of a traffic trace?
- Henceforth called the **trace complexity**
- Simple approximation: „**shuffle&compress**“
  - Remove structure by iterative ***randomization***
  - Difference of compression ***before and after*** randomization: structure

# The Trace Complexity

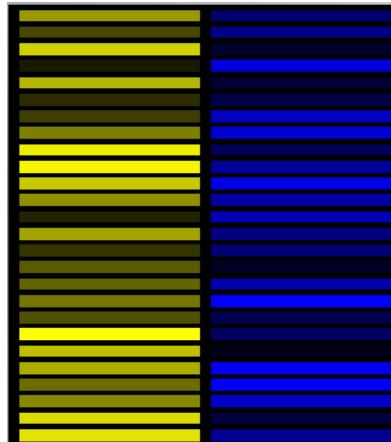
Original src-dst trace



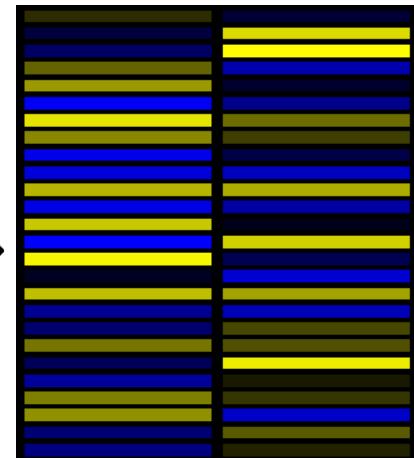
Randomize rows



Randomized columns



Uniform trace

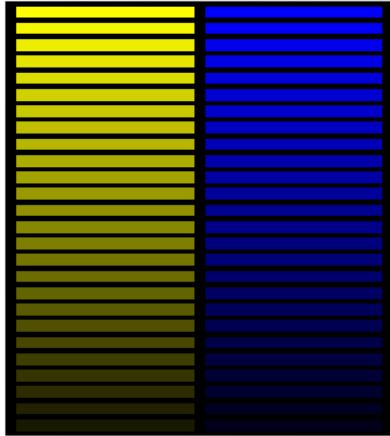


Increasing complexity (systematically randomized)

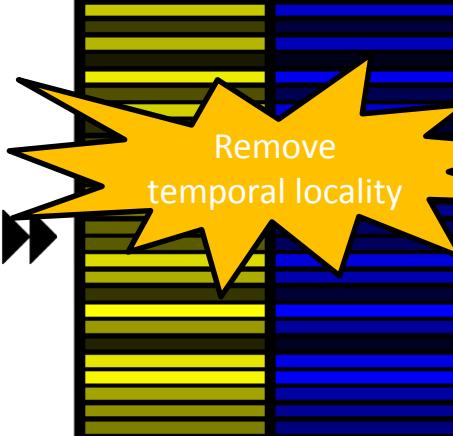
More structure (compresses better)

# The Trace Complexity

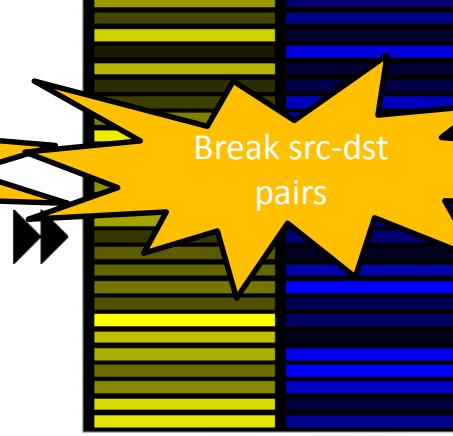
Original src-dst trace



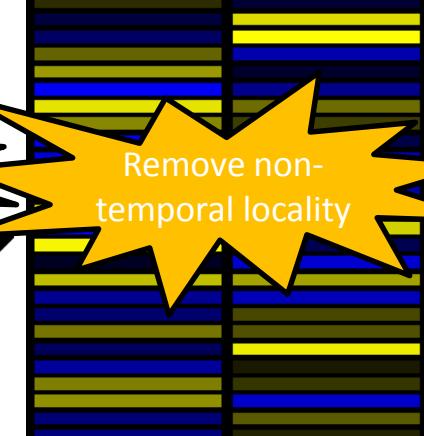
Randomize rows



Randomized columns



Uniform trace



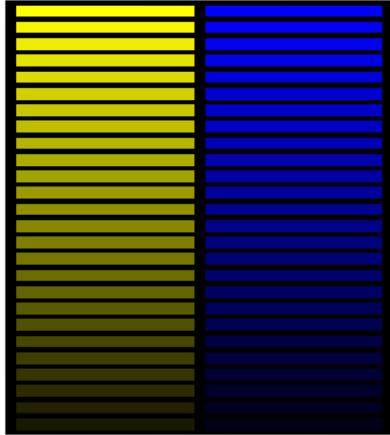
Difference in  
compression?

Difference in  
compression?

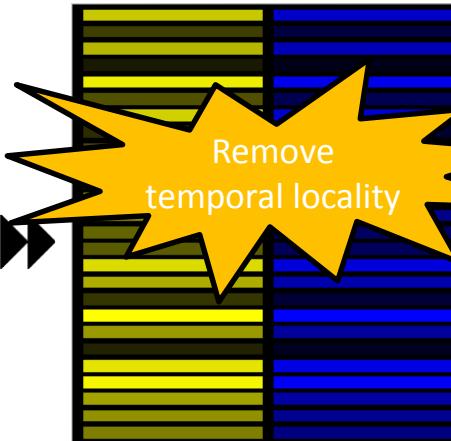
Difference in  
compression?

# The Trace Complexity

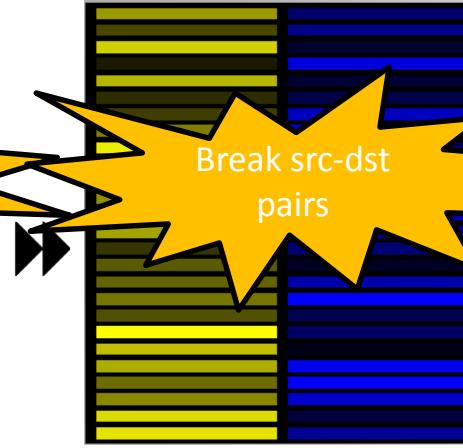
Original src-dst trace



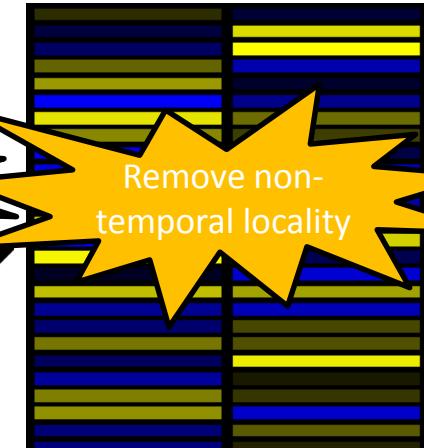
Randomize rows



Randomized columns



Uniform trace



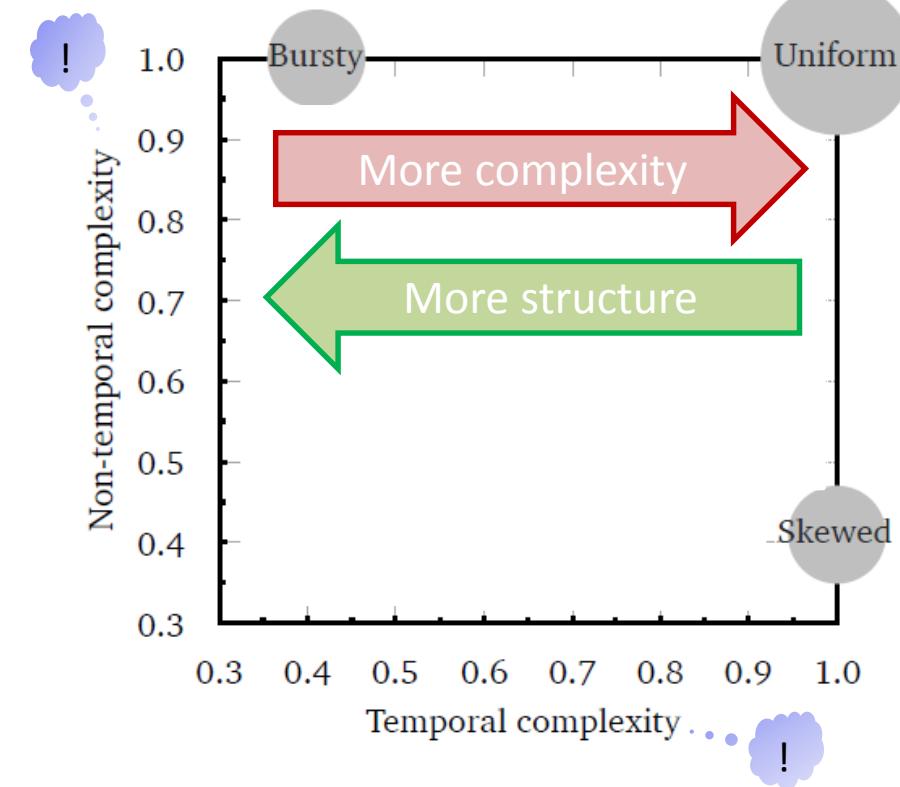
Difference in  
compression?

Difference in  
compression?

Difference in  
compression?

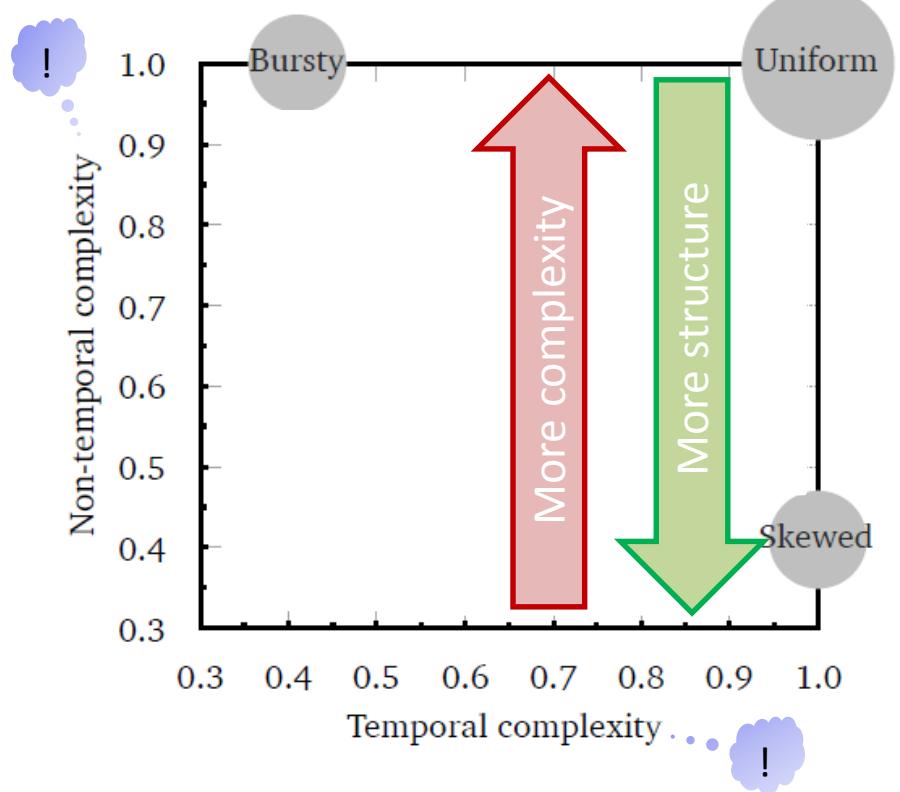
Can be used to define a „complexity map“!

# The Complexity Map



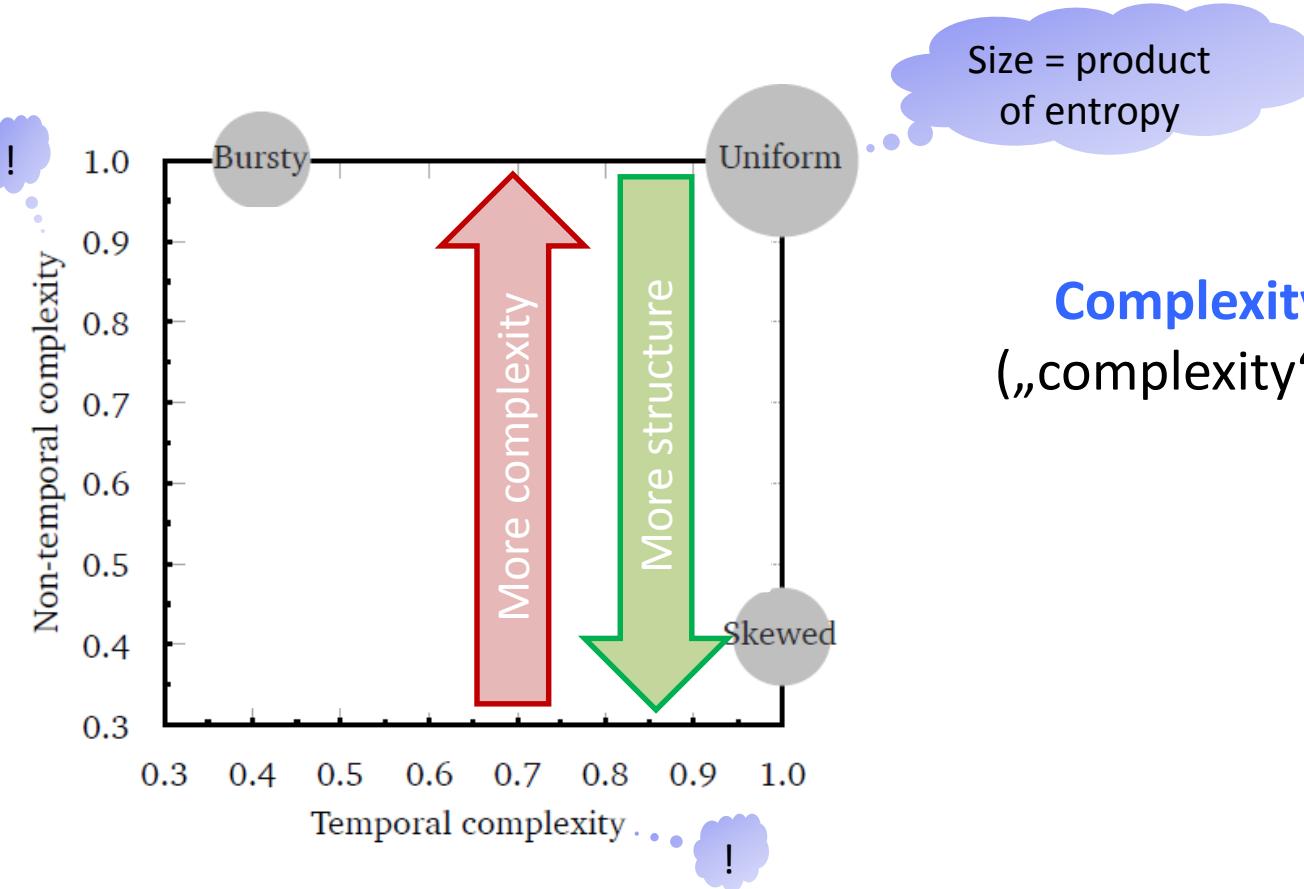
**Complexity Map:** Entropy („complexity“) of traffic traces.

# The Complexity Map



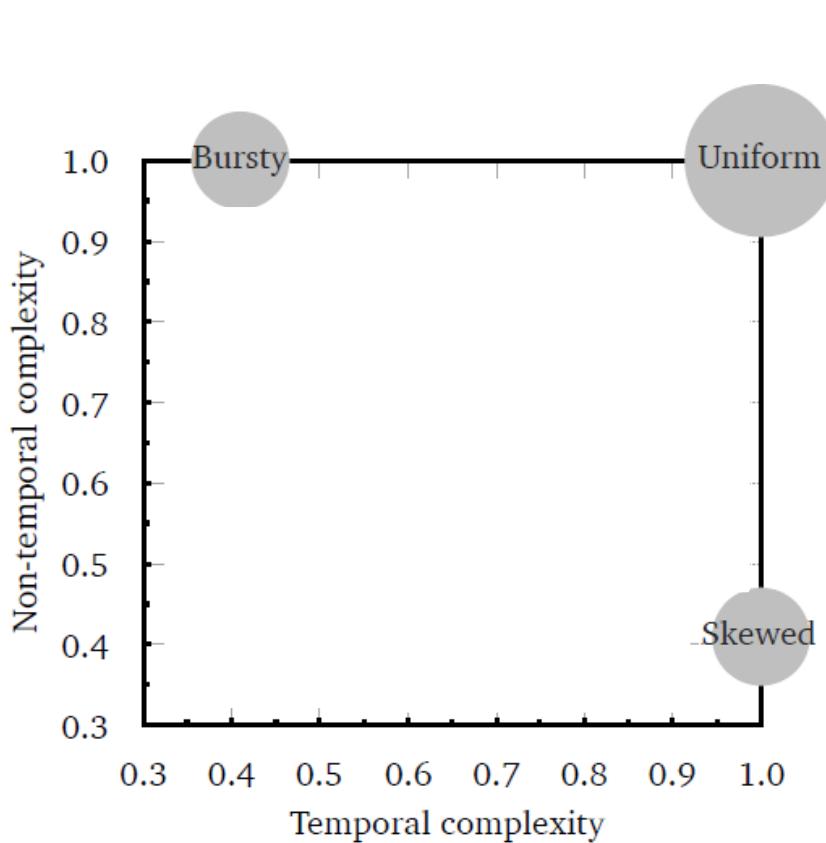
**Complexity Map:** Entropy („complexity“) of traffic traces.

# The Complexity Map



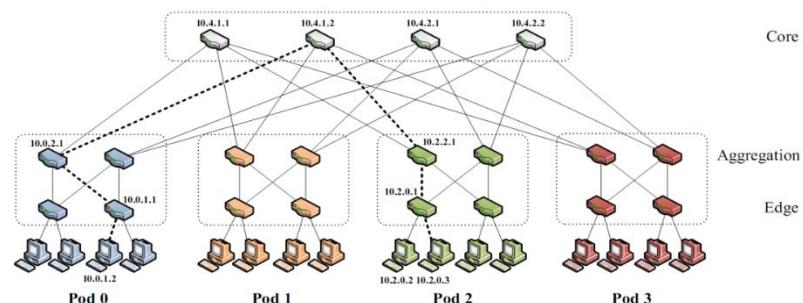
**Complexity Map:** Entropy („complexity“) of traffic traces.

# The Complexity Map

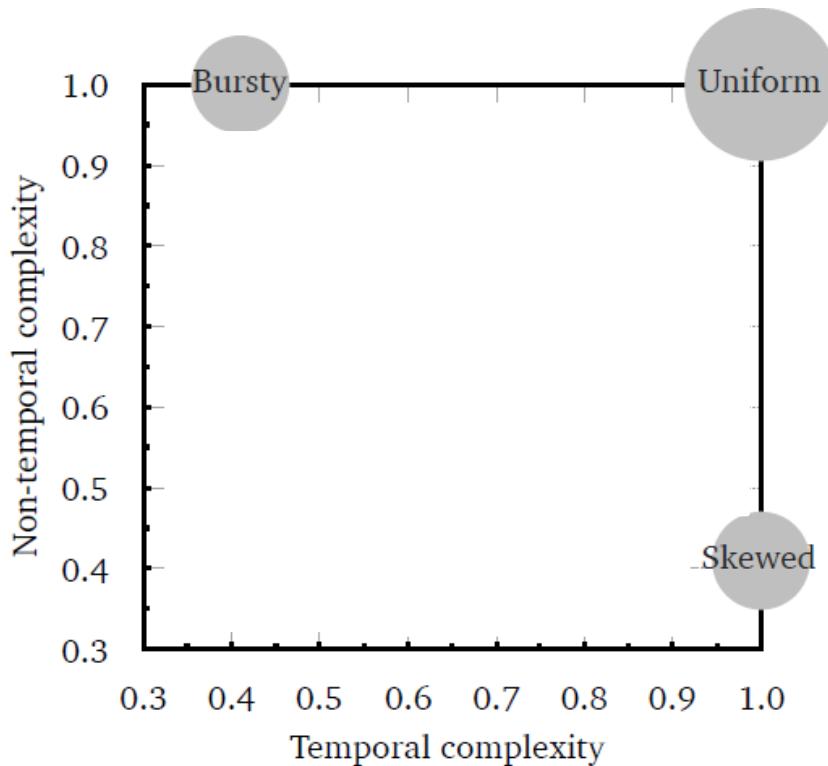


Uniform: Today's  
datacenters

- Traditional networks are optimized **for the “worst-case” (all-to-all)** communication traffic)
- Example, fat-tree topologies: provide **full bisection bandwidth**

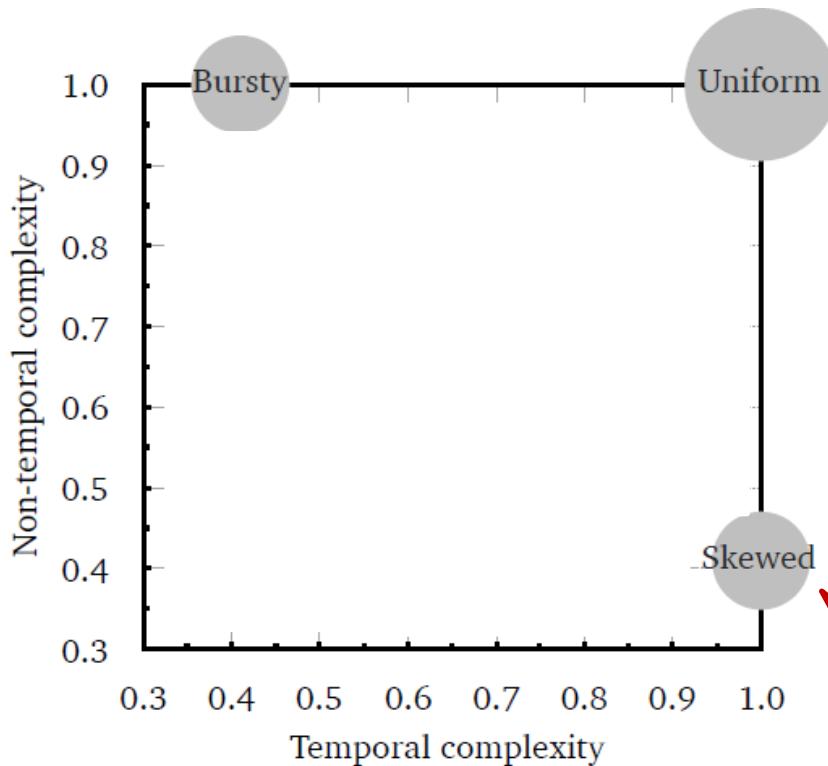


# The Complexity Map



***Good*** in the worst case ***but:***  
cannot leverage different  
**temporal** and **non-temporal**  
structures of traffic traces!

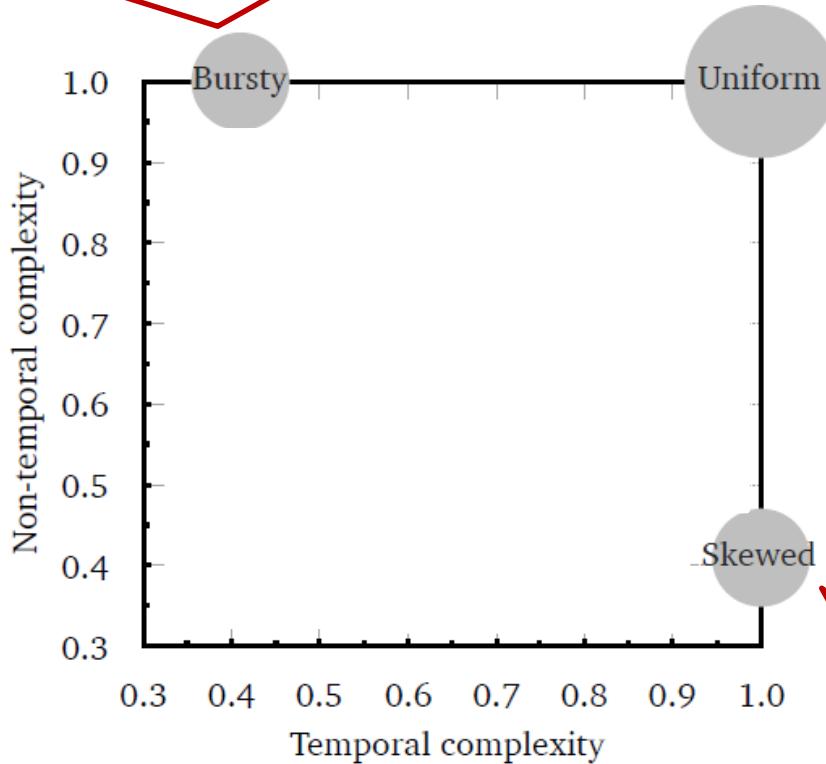
# The Complexity Map



***Good*** in the worst case ***but***:  
cannot leverage different  
**temporal** and **non-temporal**  
structures of traffic traces!

**Non-temporal** structure could  
be exploited already with ***static***  
***demand-aware networks***!

To exploit **temporal** structure,  
need ***adaptive demand-aware***  
***("self-adjusting") networks.***

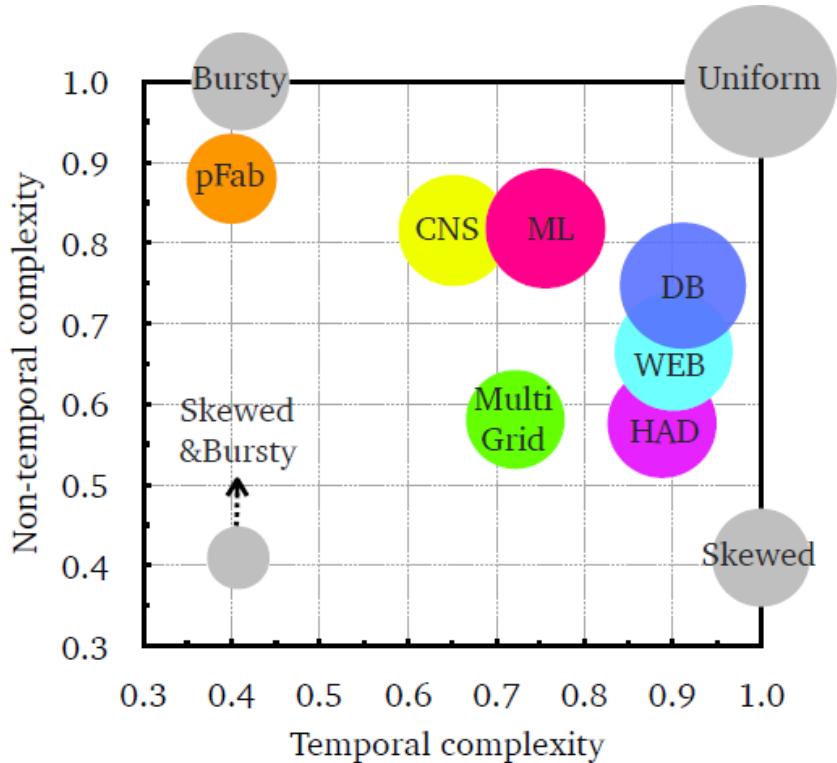


## Complexity Map

***Good*** in the worst case ***but:***  
cannot leverage different  
**temporal** and **non-temporal**  
structures of traffic traces!

**Non-temporal** structure could  
be exploited already with ***static***  
***demand-aware networks!***

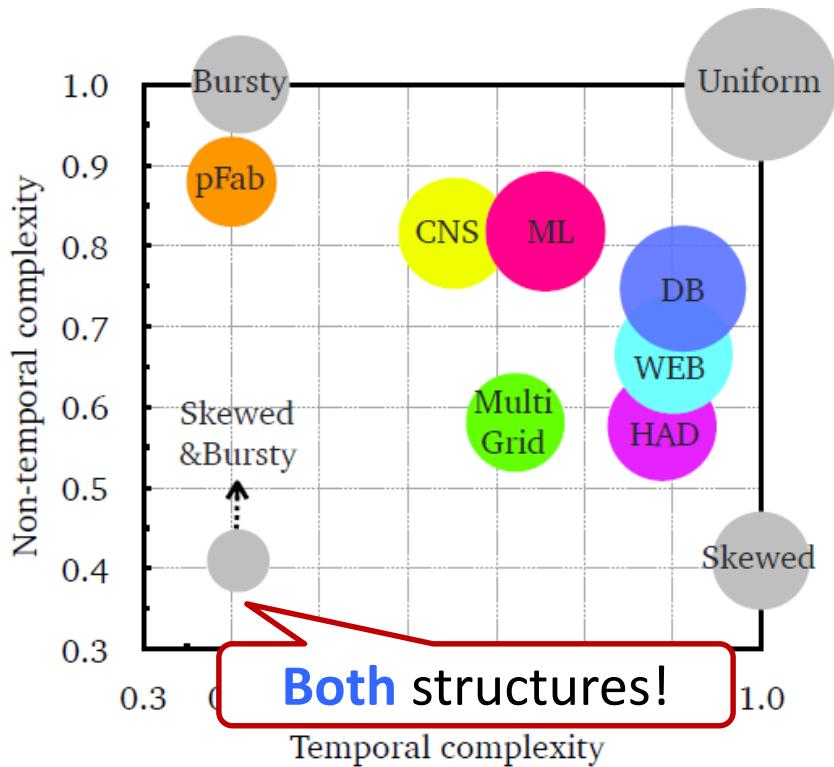
# The Complexity Map



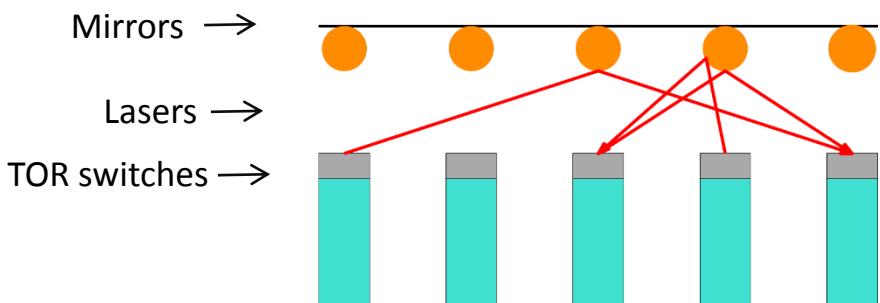
**Observation:** different applications feature quite significant (and different!) **temporal** and **non-temporal** structures.

- Facebook clusters: DB, WEB, HAD
- HPC workloads: CNS, Multigrid
- Distributed Machine Learning (ML)
- Synthetic traces like pFabric

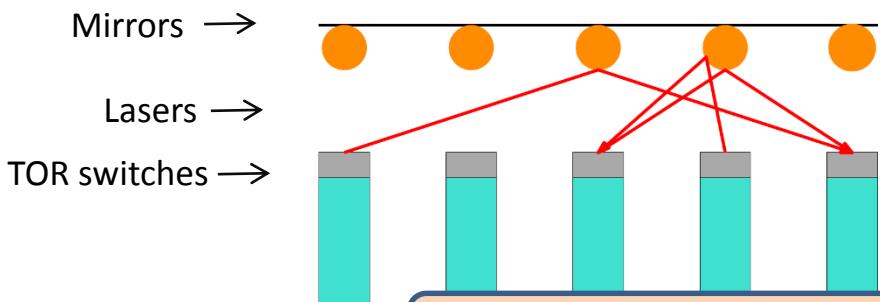
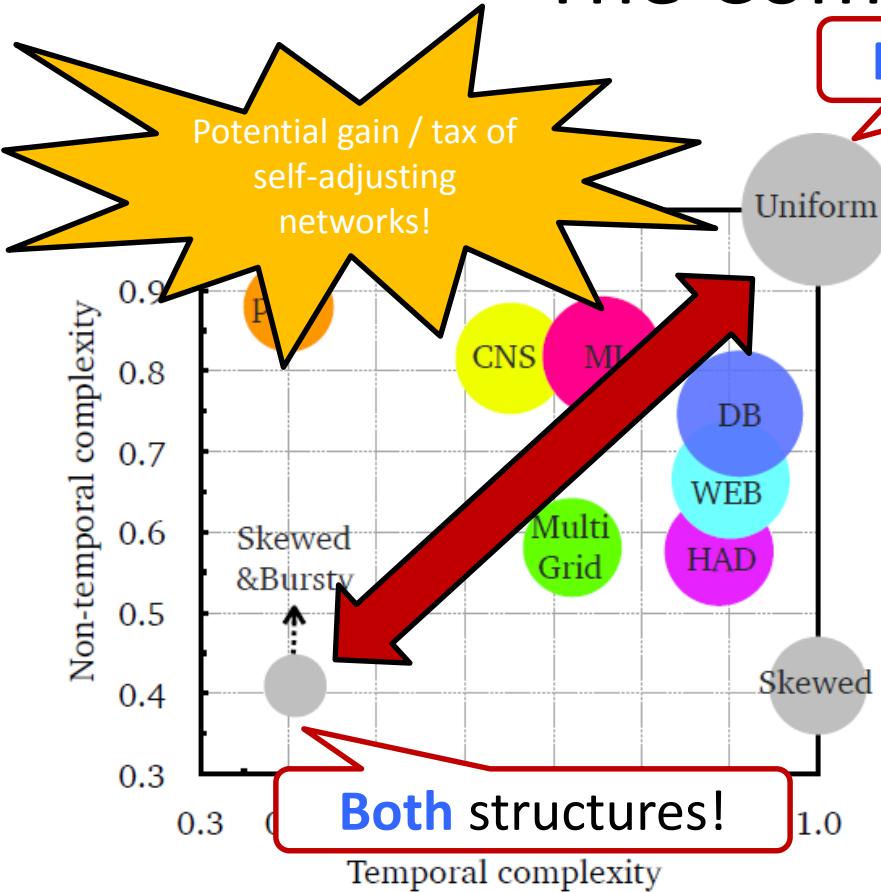
# The Complexity Map



**Goal:** Design **self-adjusting networks** which leverage **both** dimensions of structure!



# The Complexity Map



Measuring the Complexity of Packet Traces.  
Avin, Ghobadi, Griner, Schmid. **ArXiv** 2019.

---

But: How to design DANs which  
also leverage *temporal structure*?

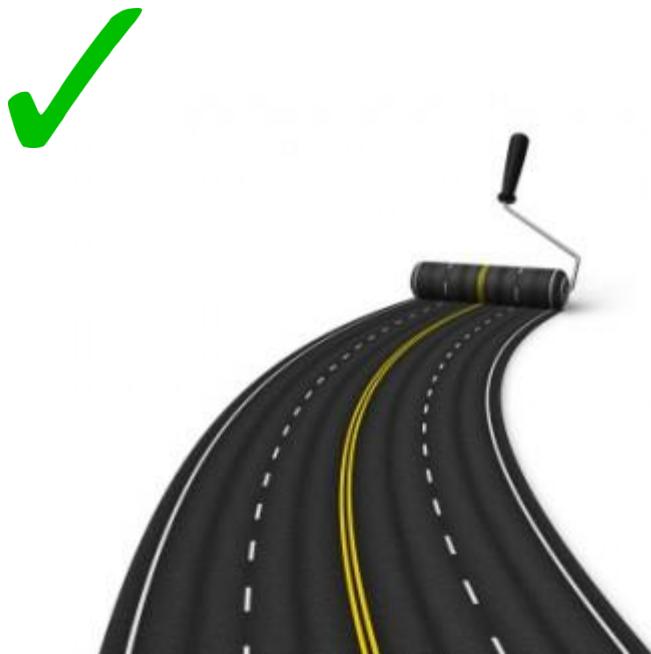
---



Inspiration from **self-adjusting  
datastructures** again!

# Roadmap

- Entropy: A metric for demand-aware networks?
  - Empirical motivation
  - A lower bound
  - Algorithms achieving entropy bounds
- From static to dynamic demand-aware networks
  - A connection to self-adjusting datastructures

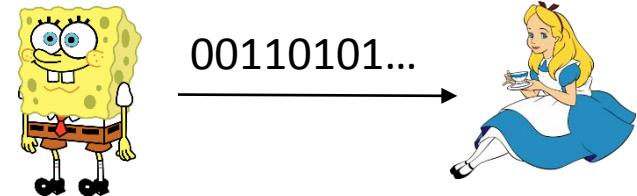


# *First: An Analogy*

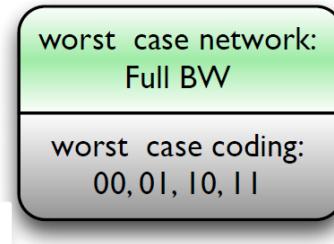
---

Static vs dynamic demand-  
aware networks!?  
**DANs vs SANs?**

# An Analogy to Coding

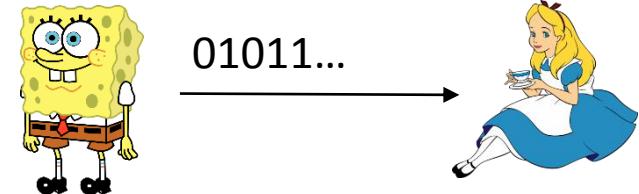


if demand **arbitrary** and **unknown**



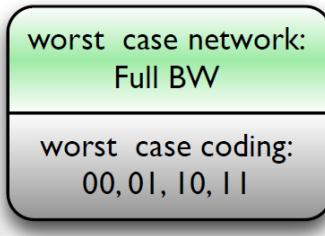
*log diameter*

*log # bits / symbol*



# An Analogy to Coding

if demand **arbitrary** and **unknown**

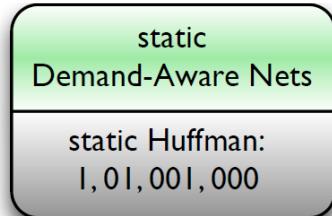


*log diameter*

*log # bits / symbol*



if demand **known** and **fixed**



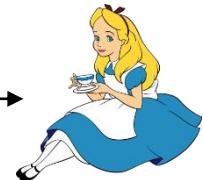
*entropy?*

*entropy / symbol*

# An Analogy to Coding

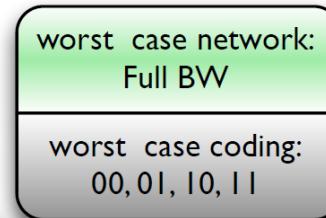


011...



if demand **arbitrary** and **unknown**

DAN!  

*log diameter*

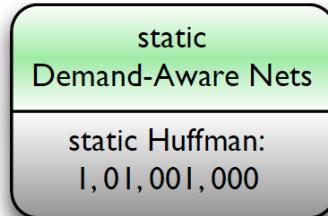
**Dynamic DANs:**  
Aka. **Self-Adjusting Networks (SANs)!**

SAN!  

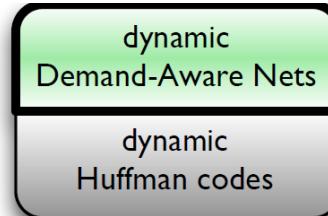

if demand **known** and **fixed**

*entropy?*

*entropy / symbol*



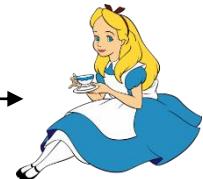
if demand **unknown** but **reconfigurable**



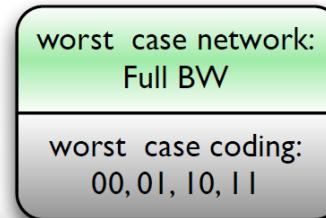
# An Analogy to Coding



011...



if demand **arbitrary** and **unknown**



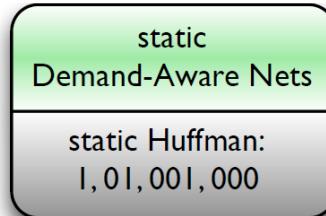
*log diameter*

*log # bits / symbol*

**Dynamic DANs:**  
Aka. **Self-Adjusting Networks (SANs)!**



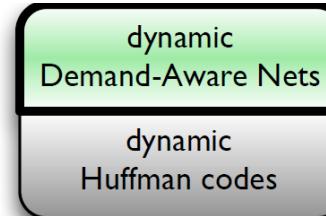
if demand **known** and **fixed**



Can exploit  
**spatial locality!**



if demand **unknown** but **reconfigurable**



Additionally exploit  
**temporal locality!**

# An Analogy to Coding



011...



if demand **arbitrary** and **unknown**

worst case network:  
Full BW

worst case coding:  
00, 01, 10, 11

*log diameter*

**Dynamic DANs:**  
Aka. **Self-Adjusting Networks** (SANs)!



if demand **know**



„Cheating“: need to know demand!



Can exploit  
**spatial locality**!

Aware Nets  
static Huffman:  
1, 01, 001, 000

*log # bits / symbol*



if demand **unknown** but **repeated**

dynamic  
Demand  
Huffman codes

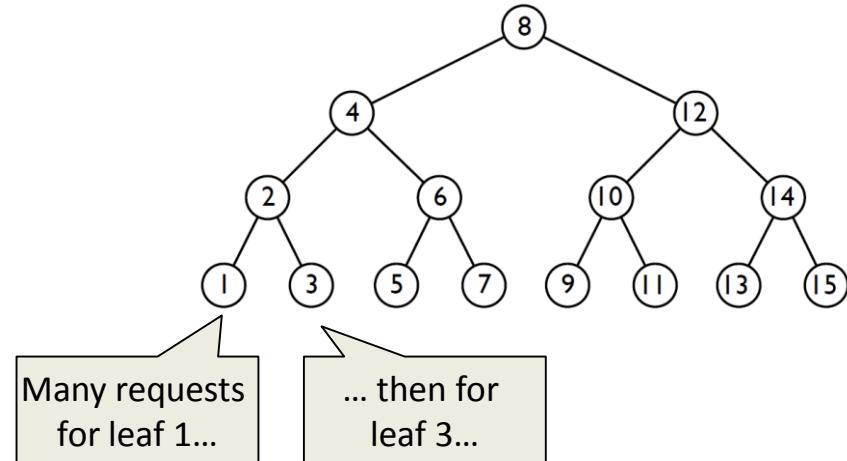
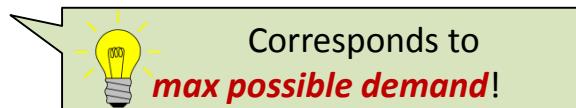
Need online algorithms!



Additionally exploit  
**temporal locality**!

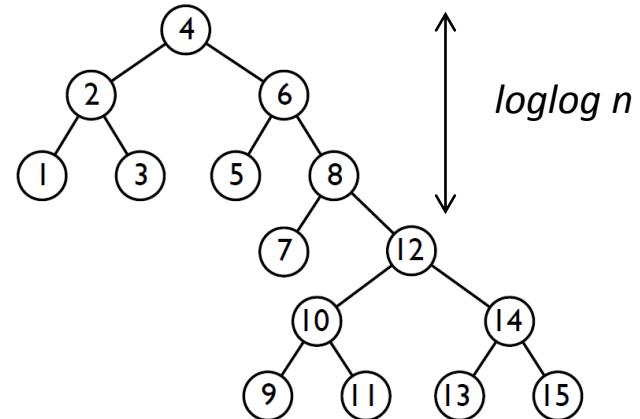
# Analogous to *Datastructures*: Oblivious...

- Traditional, **fixed** BSTs do not rely on any assumptions on the demand
- Optimize for the **worst-case**
- Example **demand**:  
 $1, \dots, 1, 3, \dots, 3, 5, \dots, 5, 7, \dots, 7, \dots, \log(n), \dots, \log(n)$   
 $\longleftrightarrow \longleftrightarrow \longleftrightarrow \longleftrightarrow \longleftrightarrow \quad \longleftrightarrow$   
*many many many many*                           *many*
- Items stored at  **$O(\log n)$**  from the root, **uniformly** and **independently** of their frequency



# ... Demand-Aware ...

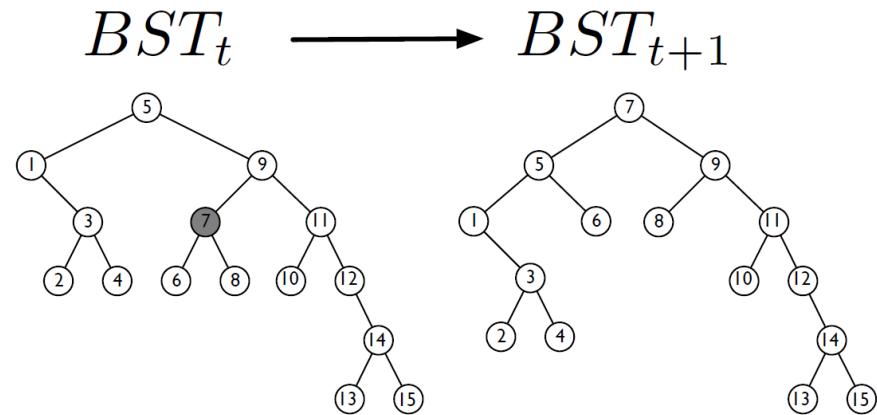
- **Demand-aware fixed** BSTs can take advantage of *spatial locality* of the demand
- E.g.: place frequently accessed elements close to the root
- E.g., **Knuth/Mehlhorn/Tarjan** trees
- Recall example **demand**:  
 $1, \dots, 1, 3, \dots, 3, 5, \dots, 5, 7, \dots, 7, \dots, \log(n), \dots, \log(n)$ 
  - Amortized cost  $O(\log \log n)$



 Amortized cost corresponds  
to *empirical entropy of demand!*

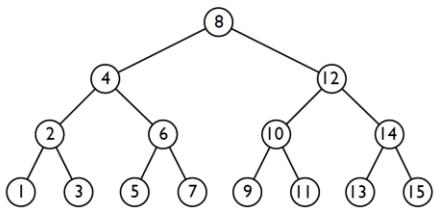
# ... Self-Adjusting!

- Demand-aware reconfigurable BSTs can additionally take advantage of *temporal locality*
- By moving accessed element to the root: amortized cost is *constant*, i.e.,  $O(1)$ 
  - Recall example demand:  
 $1, \dots, 1, 3, \dots, 3, 5, \dots, 5, 7, \dots, 7, \dots, \log(n), \dots, \log(n)$

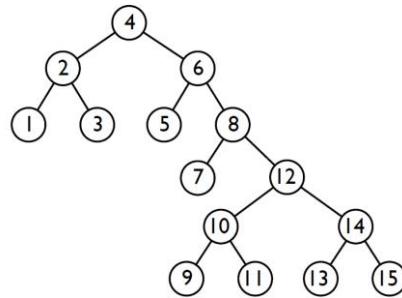


# Datastructures

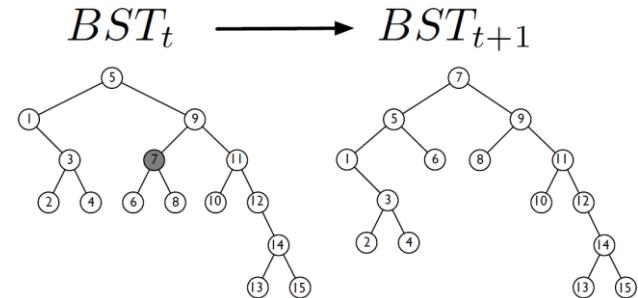
Oblivious



Demand-Aware



Self-Adjusting



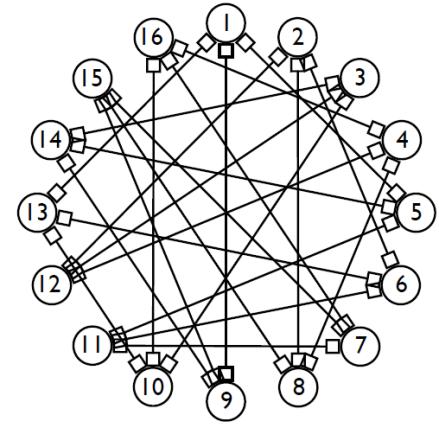
Lookup  
 $O(\log n)$

Exploit **spatial locality**:  
*empirical entropy  $O(\log \log n)$*

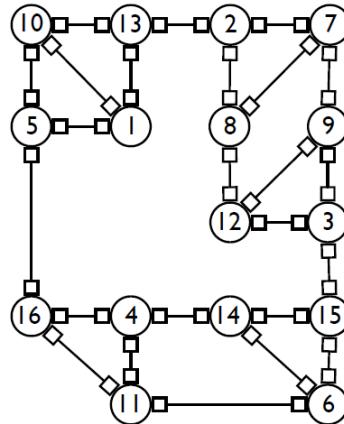
Exploit **temporal locality** as well:  
 $O(1)$

# Analogously for Networks

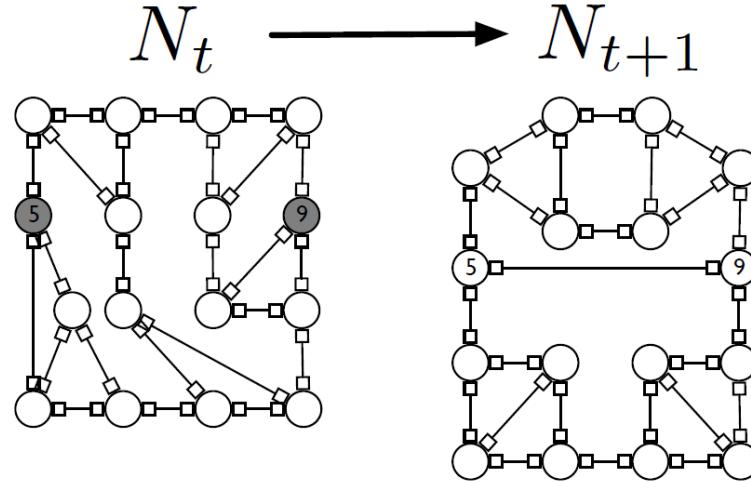
Oblivious



DAN



SAN



Const degree  
(e.g., **expander**):  
route lengths  **$O(\log n)$**

Exploit **spatial locality**

Exploit **temporal locality** as well

# Now: Design of Self-Adjusting Networks (SANs)

---



Inspiration from **self-adjusting  
datastructures** again!

---

What's the model?

---

---

# What's the model?

---

Again: *it depends...* ☺

# The Problem Input

A **sequence**  $\sigma = (u_1, v_1), (u_2, v_2), (u_3, v_3) \dots$

chosen arbitrarily

Chosen i.i.d. from initially  
unknown fixed distribution

# The Problem Input

A **sequence**  $\sigma = (u_1, v_1), (u_2, v_2), (u_3, v_3) \dots$

revealed online

given offline

chosen arbitrarily

Chosen i.i.d. from initially  
unknown fixed distribution

# The Problem Input

A **sequence**  $\sigma = (u_1, v_1), (u_2, v_2), (u_3, v_3) \dots$

revealed online

given offline

chosen arbitrarily

Chosen i.i.d. from initially  
unknown fixed distribution

*Other options:* sequences of *snapshots*,  
generated according to *Markov process*, ...

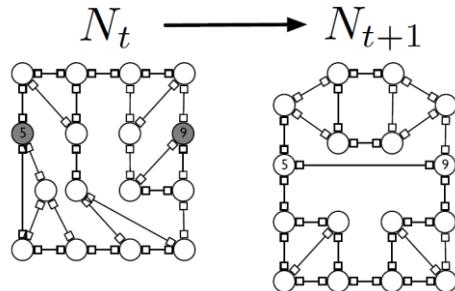
---

# What's the objective? Metric?

---

Also here: *it depends...* ☺

# A Cost-Benefit Tradeoff



Short routes  
High reconfiguration cost



Low reconfiguration cost  
Long routes

Basic question:

***How often*** to reconfigure?

# A Metric

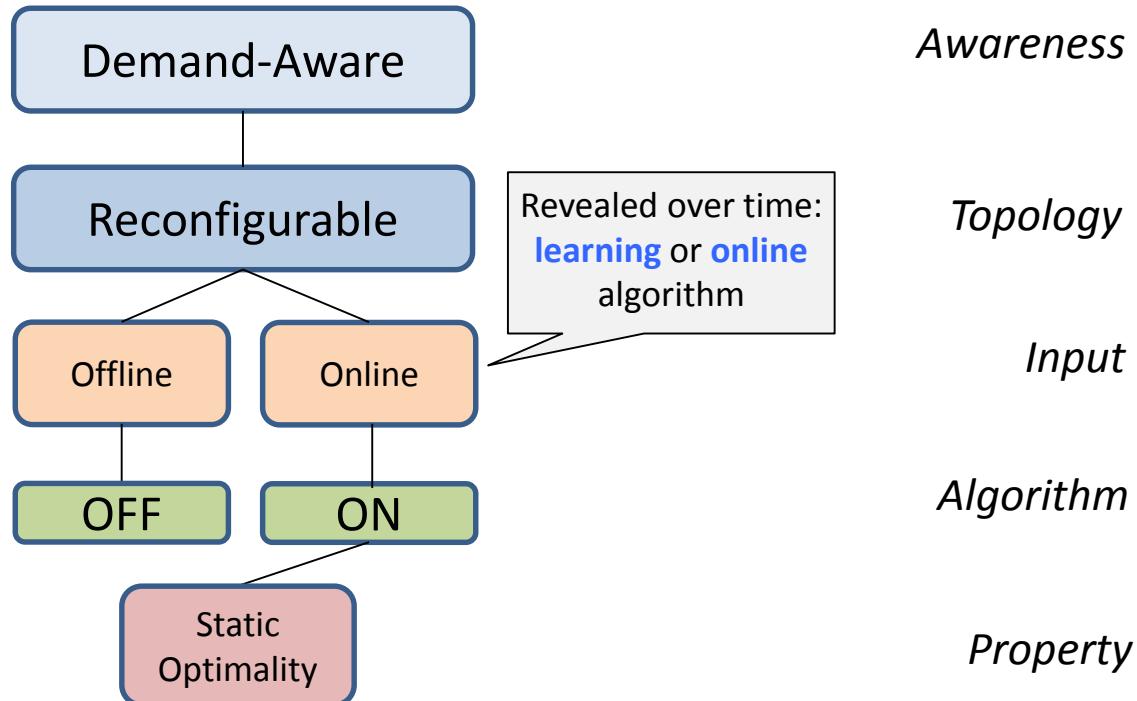
Entropy of the demand again...



... but now **entropy rate** (entropy over time)!

# A Taxonomy: Reconfigurable Networks

**Static Optimality:**  
**“Not worse than static**  
which knows demand  
ahead of time!”  
 $\rho = \text{Cost(ON)}/\text{Cost(STAT*)}$   
is constant.

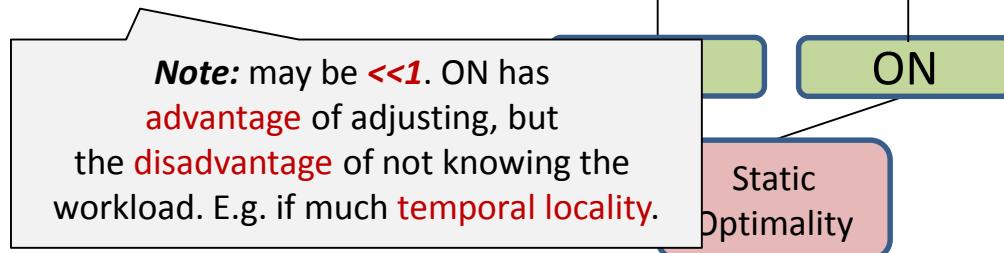


# A Taxonomy: Reconfigurable Networks

**Static Optimality:**

**"Not worse than static**  
which knows demand  
ahead of time!"

$\rho = \text{Cost(ON)}/\text{Cost(STAT*)}$   
is constant.



*Awareness*

Revealed over time:  
**learning or online**  
algorithm

*Topology*

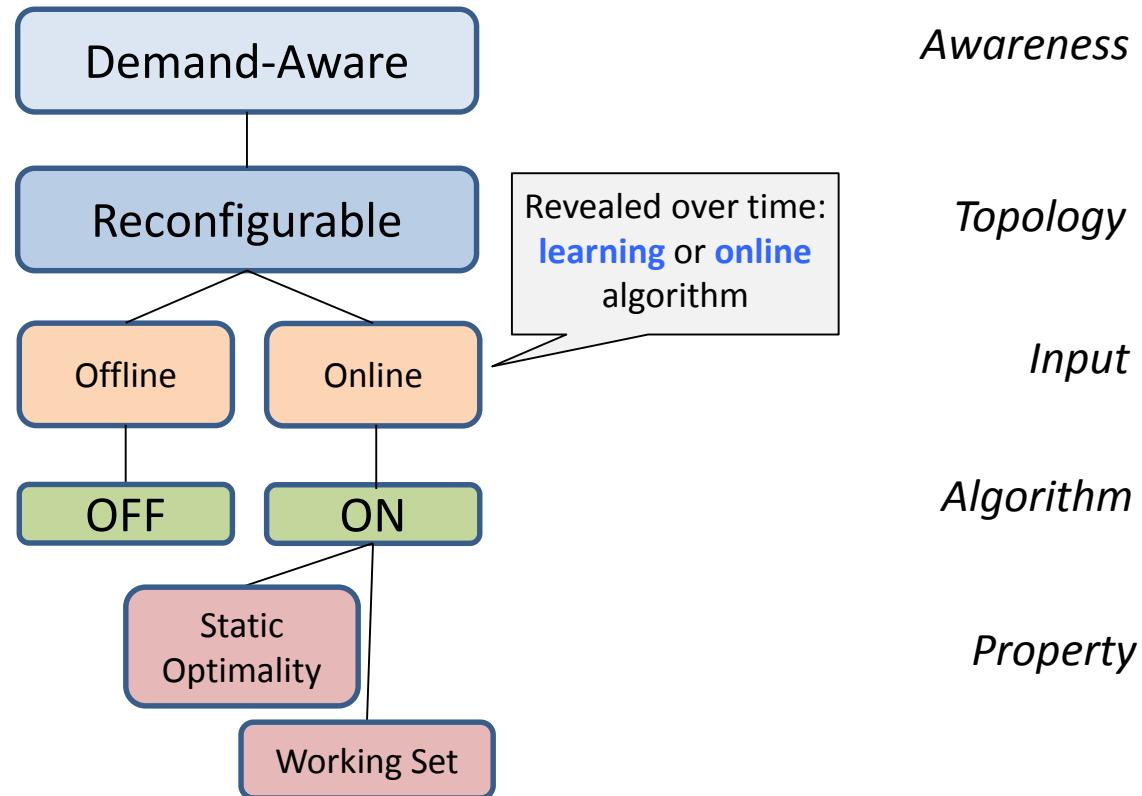
*Input*

*Algorithm*

*Property*

# A Taxonomy: Reconfigurable Networks

**Working Set Property:**  
“*Topological distance* between nodes  
*proportional to how recently they communicated!*”



# A Taxonomy: Reconfigurable Networks

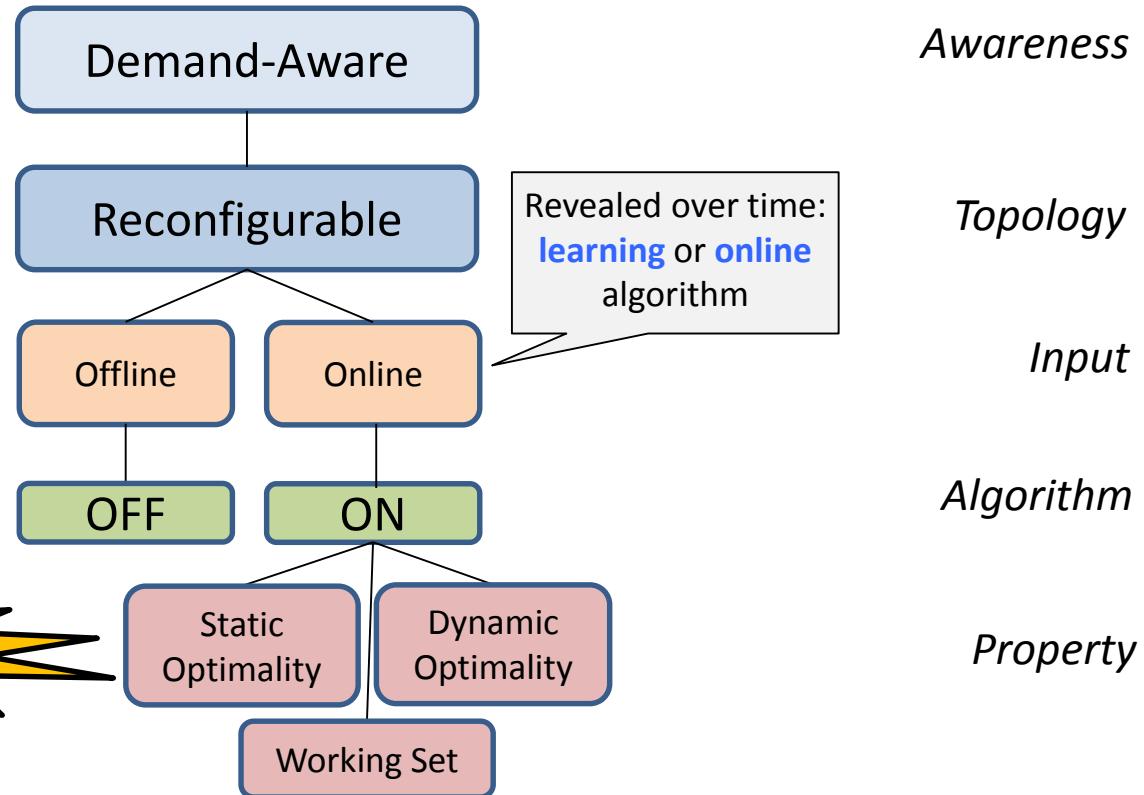
**Dynamic Optimality:**

*“No worse than an offline* algorithm which *knows the sequence!*”

$\rho = \text{Cost(ON)}/\text{Cost(OFF*)}$   
is constant.

Always  $\geq 1$ .

The holy grail!



---

# Algorithms for Self-Adjusting Networks

---

# Algorithms for Self-Adjusting Networks

---



Let us start with **trees** again:  
Self-adjusting tree?

# Algorithms for Self-Adjusting Networks



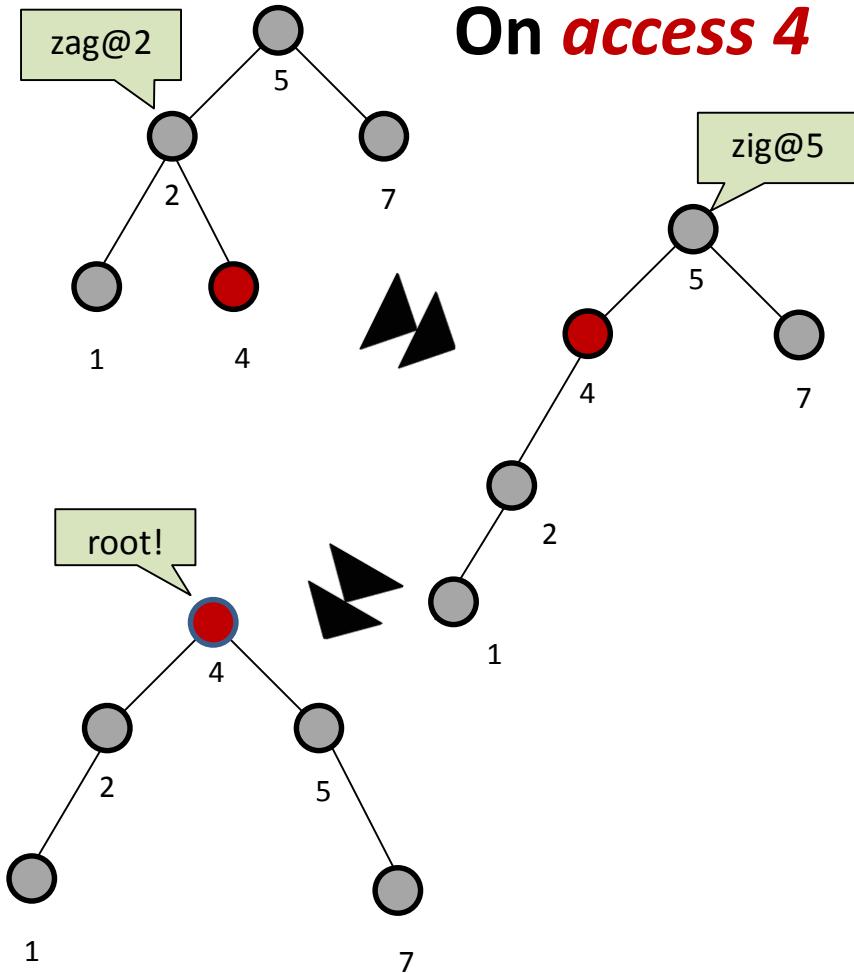
Let us start with **trees** again:  
Self-adjusting tree?



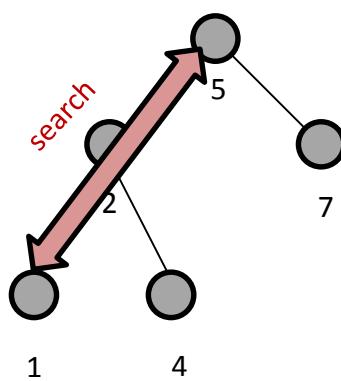
Use **self-adjusting BST!**

# Recall: Splay Tree

- A Binary Search Tree (**BST**)
- Inspired by “**move-to-front**”: move **to root!**
- Self-adjustment: **zig**, **zigzag**, **zigzag**
  - Maintains **search property**
- Many nice properties
  - **Static optimality**, **working set**, (static,dynamic) **fingers**, ...

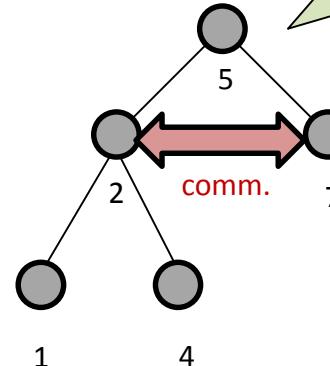


# A Simple Idea: Generalize Splay Tree To *SplayNet*



Splay Tree

vs

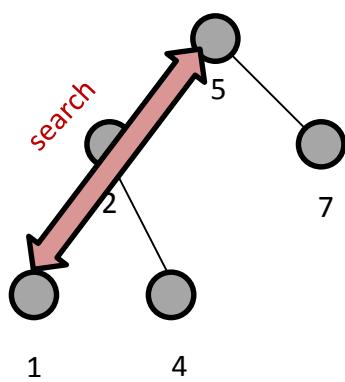


SplayNet

BST is nice for networks:  
*local (greedy) search!*

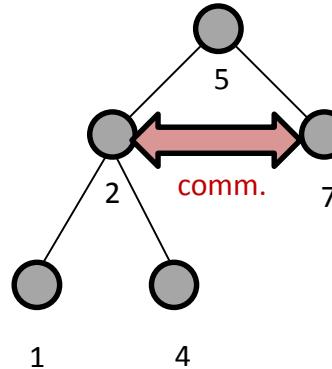
# A Simple Idea: Generalize Splay Tree To *SplayNet*

*But how?*



**Splay Tree**

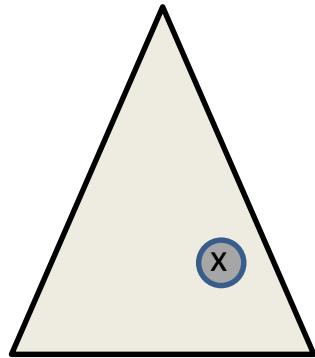
vs



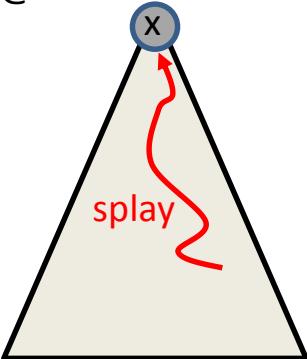
**SplayNet**

# SplayNet: A Simple Idea

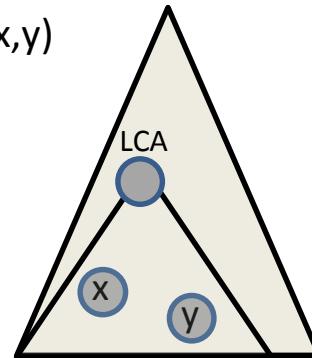
@t: access x



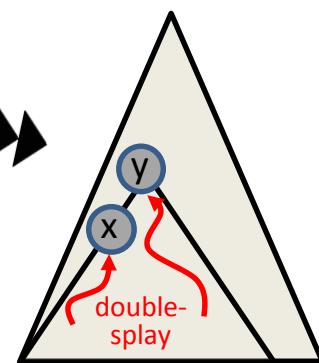
@t+1



@t: comm  
(x,y)



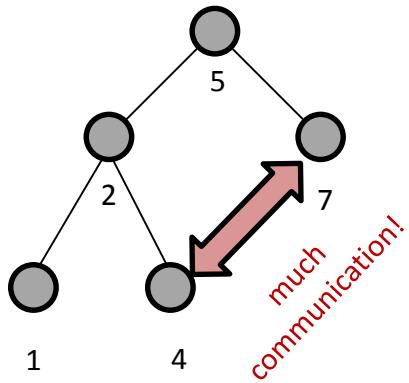
@t+1



**Splay Tree**

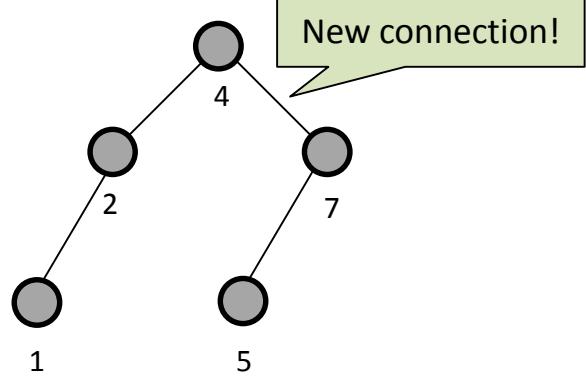
**SplayNet**

# Example



$t=1$

adjust



$t=2$

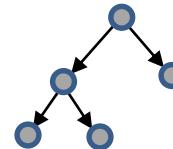
**Challenges:** How to minimize reconfigurations?

How to keep network locally routable?

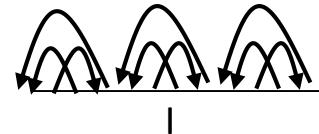
# Properties of SplayNets

- **Statically optimal** if demand comes from a *product distribution*
  - Product distribution: entropy equals conditional entropy, i.e.,  $H(X)+H(Y)=H(X|Y)+H(Y|X)$
- Converges to optimal static topology in
  - **Multicast scenario**: requests come from a *binary tree* as well
  - **Cluster scenario**: communication only *within* interval
  - **Laminated scenario** : communication is „*non-crossing matching*“

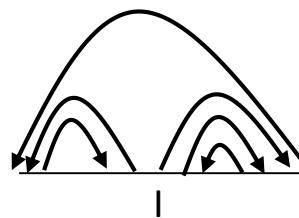
Multicast Scenario



Cluster Scenario



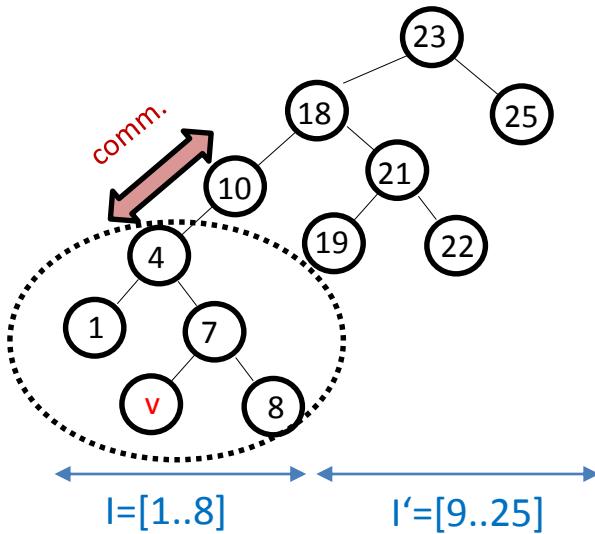
Laminated Scenario



# Remark: Static SplayNet

**Theorem:** Optimal static SplayNet can be computed in polynomial-time (**dynamic programming**)

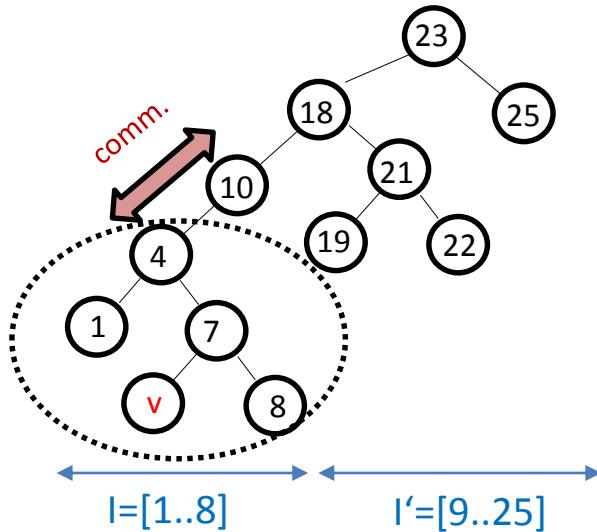
- Unlike unordered tree?



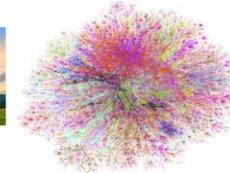
# Remark: Static SplayNet

**Theorem: Optimal static SplayNet can be computed in polynomial-time ([dynamic programming](#))**

- Unlike unordered tree?

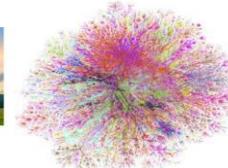


# Algorithms for Self-Adjusting Networks II



From trees to networks!

# Algorithms for Self-Adjusting Networks II

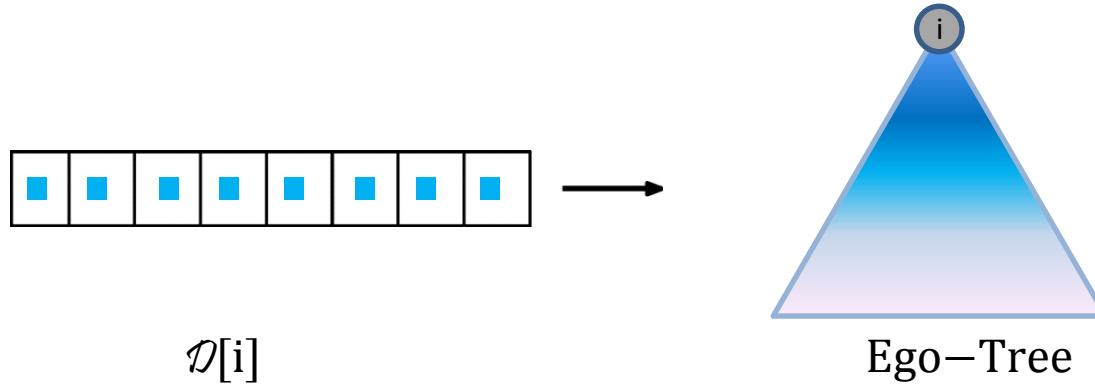


From trees to networks!

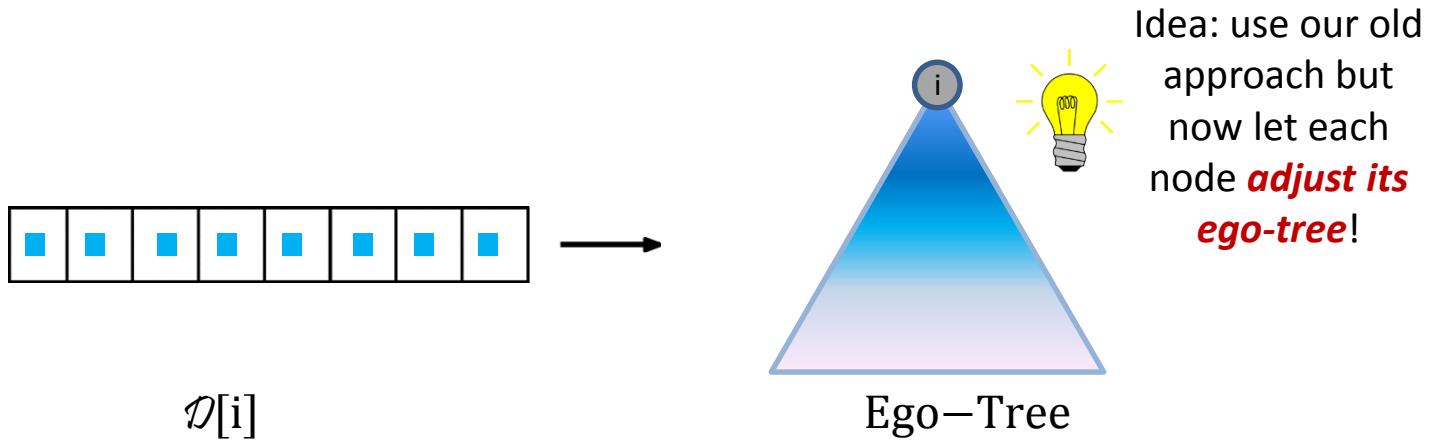


Ego-trees strike back!

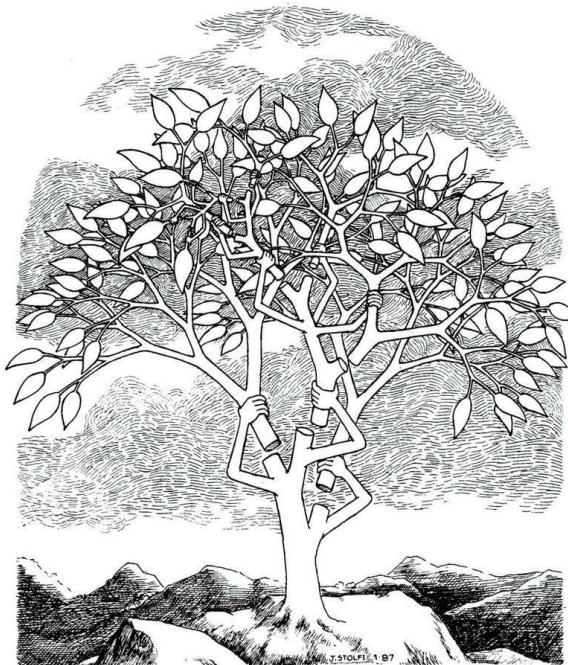
# Total Recall: Ego-Trees!



# Total Recall: Ego-Trees!

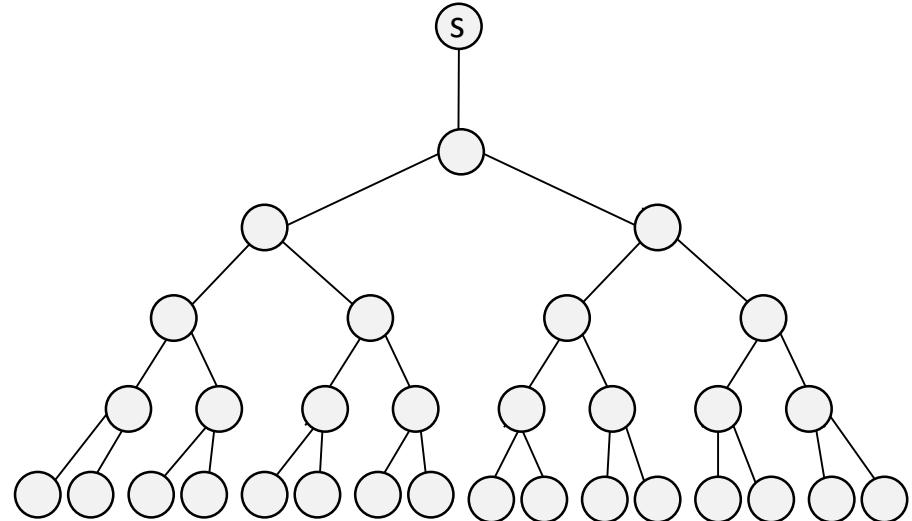


# A Dynamic Ego-Tree: Splay Tree



# An Alternative Dynamic Ego-Tree: Push-Down Tree

- **Push-down tree:** a self-adjusting complete tree
- ***Dynamically optimal***
- Not ordered: requires **a map**

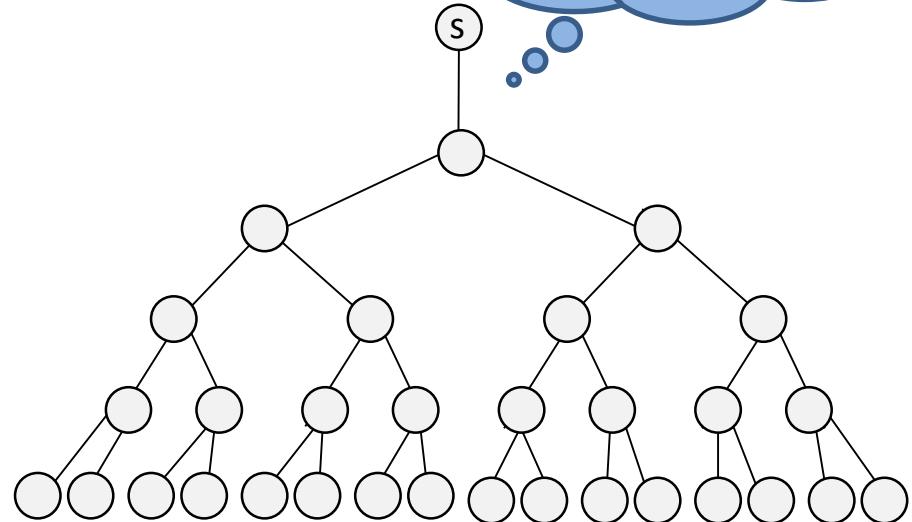


# An Alternative Dynamic Ego-Tree: Push-Down Tree



- **Push-down tree:** a self-adjusting complete tree
- ***Dynamically optimal***
- Not ordered: requires **a map**

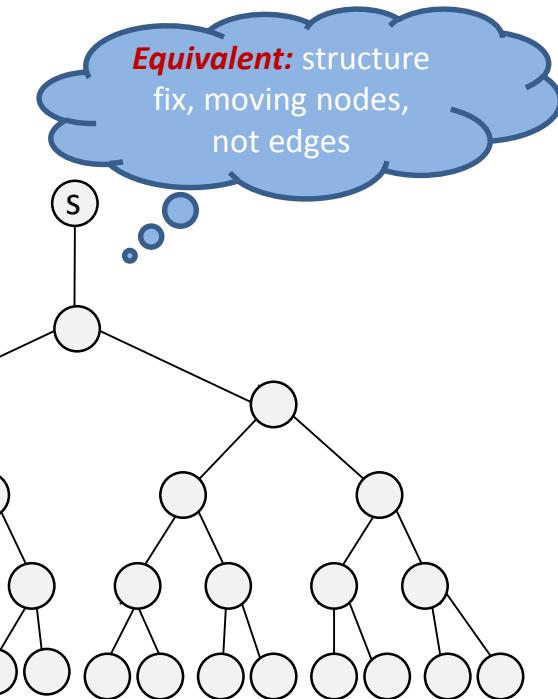
A blue cloud-shaped graphic with a white border. Inside, the word "Equivalent:" is written in red, followed by "structure fix, moving nodes, not edges" in black.



# An Alternative Dynamic Ego-Tree: Push-Down Tree



- **Push-down tree:** a self-adjusting complete tree
- **Dynamically optimal**
- Not ordered: requires **a map**

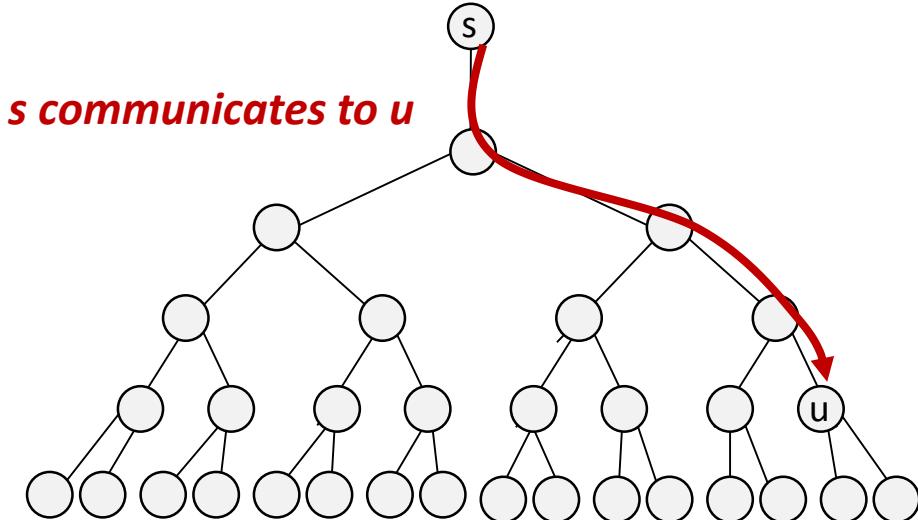


A useful dynamic property: **Most-Recently Used (MRU)**!

Similar to **Working Set Property**: more recent communication Partners closer to source.

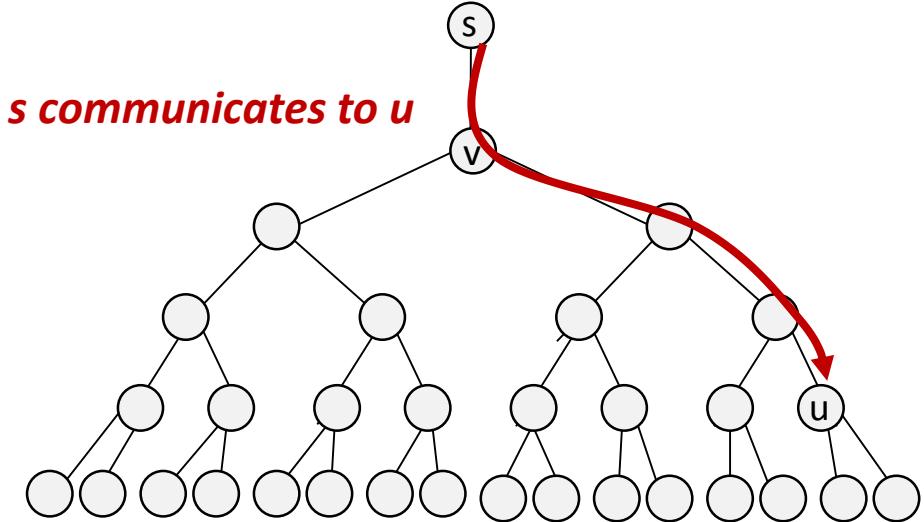
# An Alternative Dynamic Ego-Tree: Push-Down Tree

- **Push-down tree:** a self-adjusting complete tree
- ***Dynamically optimal***
- Not ordered: requires **a map**



# An Alternative Dynamic Ego-Tree: Push-Down Tree

- **Push-down tree:** a self-adjusting complete tree
- **Dynamically optimal**
- Not ordered: requires **a map**



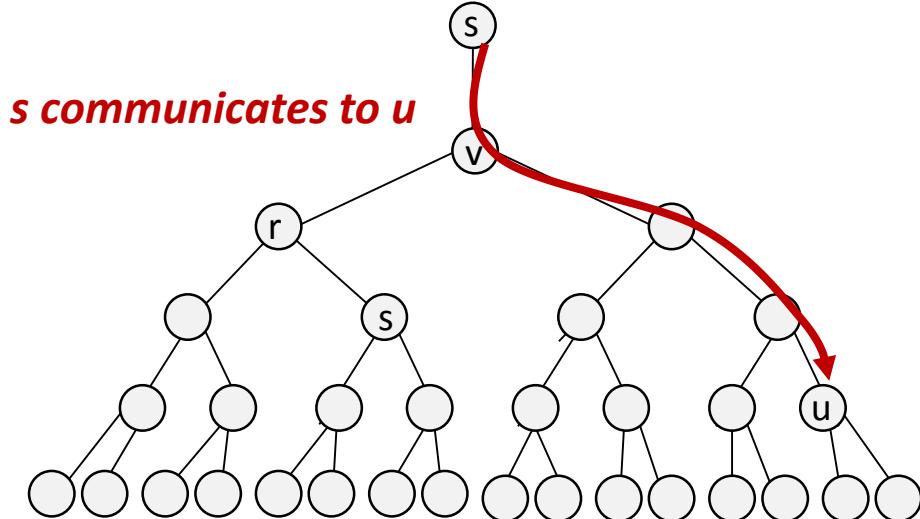
*Strict MRU requires: move *u* to root! But how? Cannot swap with *v*: *v* no longer MRU!*

# An Alternative Dynamic Ego-Tree: Push-Down Tree

- **Push-down tree:** a self-adjusting complete tree
- **Dynamically optimal**
- Not ordered: requires **a map**



*Strict MRU requires: move u to root! But how? Cannot swap with v: v no longer MRU!*



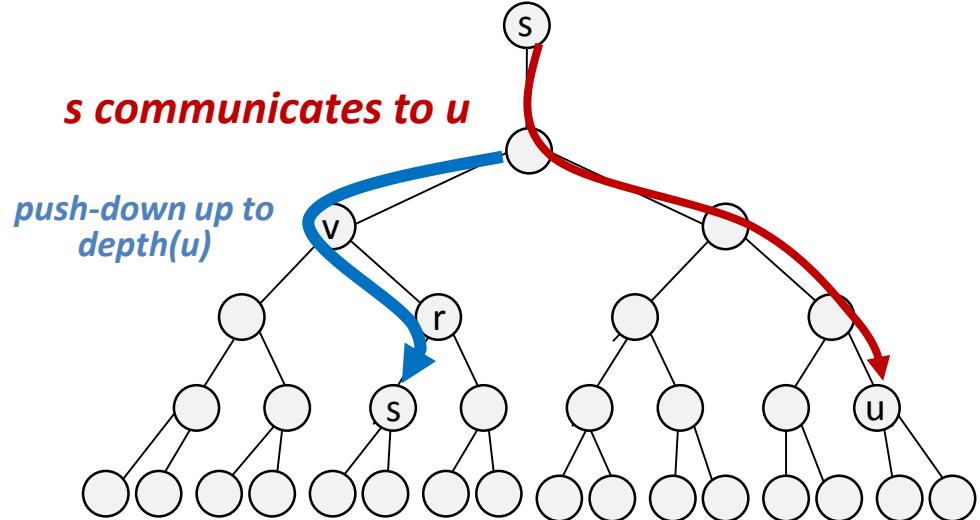
*Idea: Push v down, in a balanced manner, up to depth(u): left-right-left-right („rotate-push“)*

# An Alternative Dynamic Ego-Tree: Push-Down Tree

- **Push-down tree:** a self-adjusting complete tree
- **Dynamically optimal**
- Not ordered: requires **a map**



*Strict MRU requires: move u to root! But how? Cannot swap with v: v no longer MRU!*



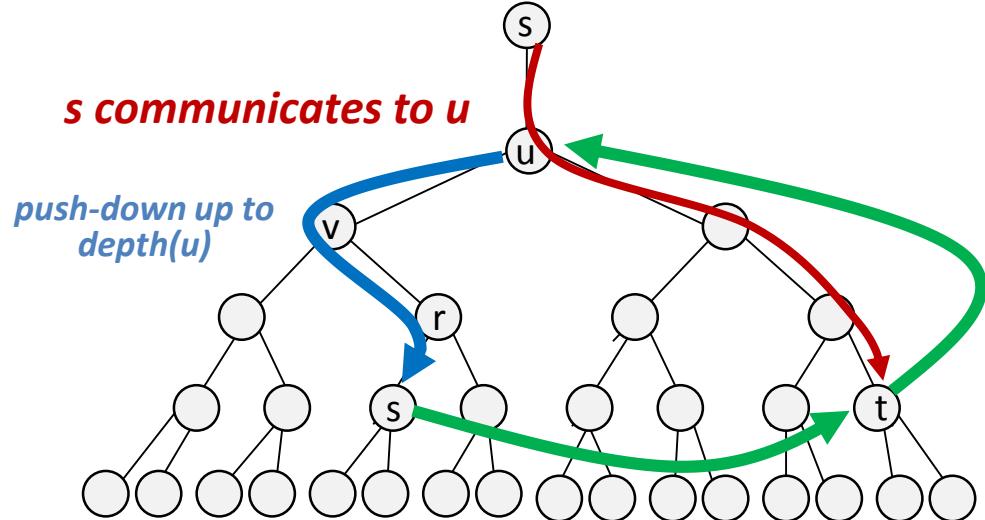
*Idea: Push v down, in a balanced manner, up to depth(u): left-right-left-right („rotate-push“)*

# An Alternative Dynamic Ego-Tree: Push-Down Tree

- **Push-down tree:** a self-adjusting complete tree
- **Dynamically optimal**
- Not ordered: requires **a map**

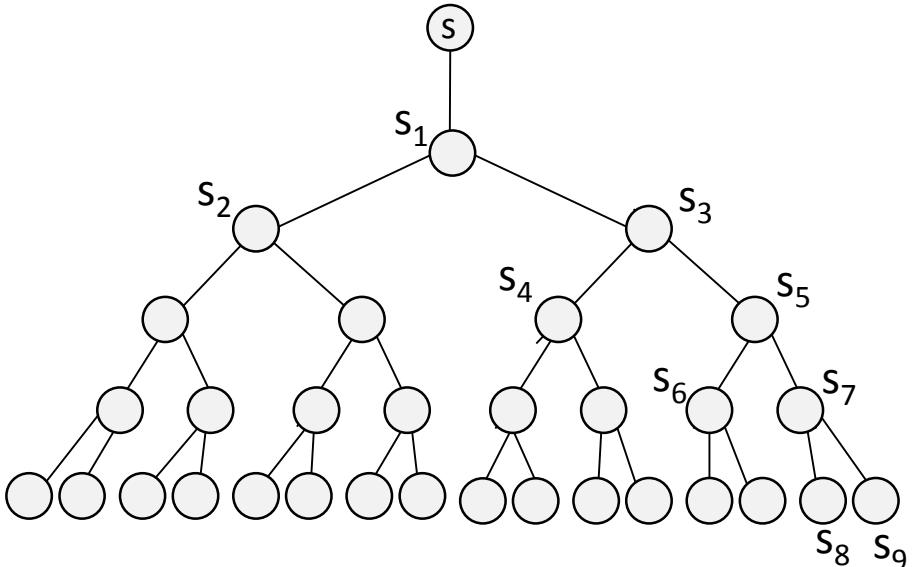


*Then: promote  $u$  to available root, and  
 $t$  to  $u$ : at original depth!*

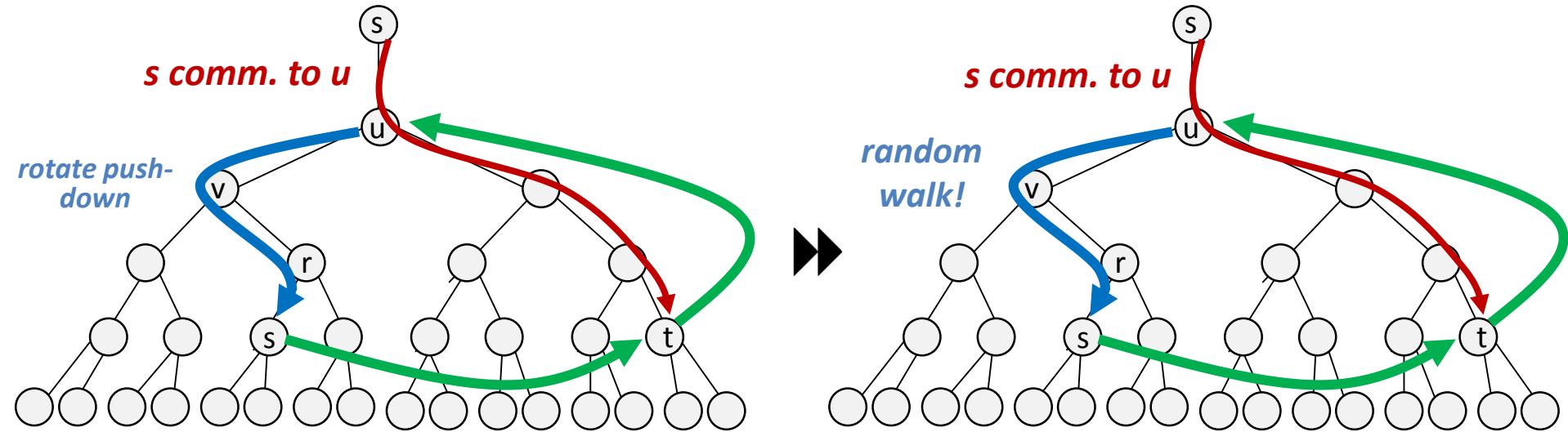


# Remarks

- Unfortunately, alternating push-down does ***not maintain MRU*** (working set) property
- Tree can ***degrade***, e.g.: sequence of requests from level 4,1,2,1,3,1,4,1

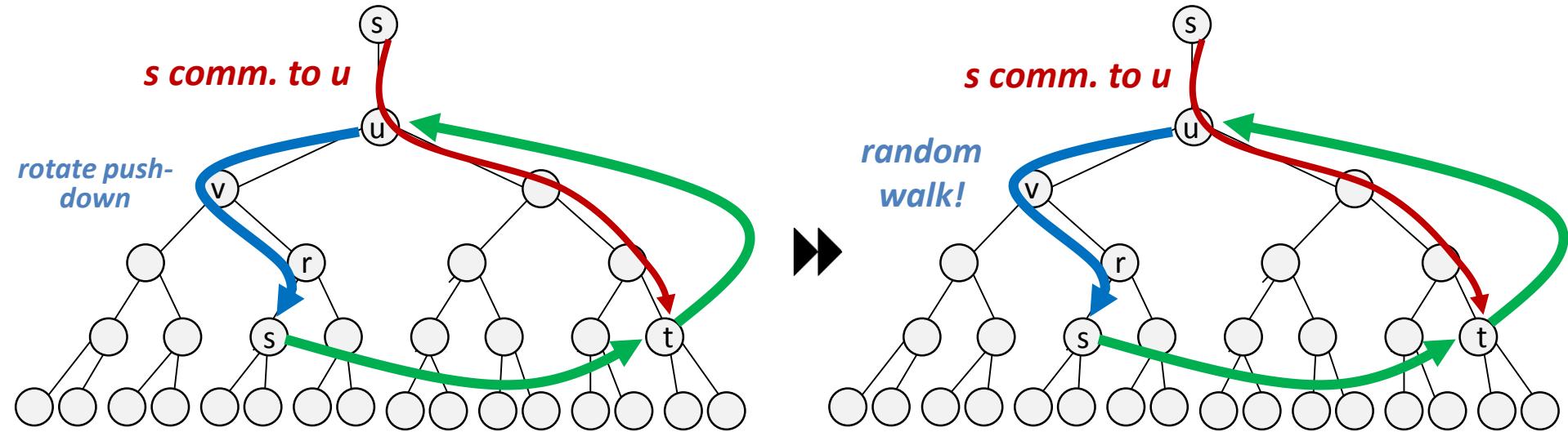


# Solution: Random Walk



# *At least maintains approximate working set / MRU!*

# Solution: Random Walk



*At least maintaining the working set*

Push-Down Trees: Optimal Self-Adjusting Complete Trees  
Chen Avin, Kaushik Mondal, and Stefan Schmid.  
ArXiv Technical Report, July 2018.

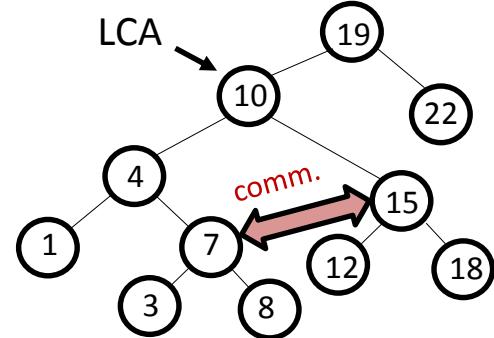
---

## Remark 1: Decentralized Algorithms

---

# A “Simple” Decentralized Solution: Distributed SplayNet (*DiSplayNet*)

- SplayNet attractive: ordered BST supports **local routing**
  - Nodes **maintain three ranges**: interval of left subtree, right subtree, upward
- If communicate (frequently): **double-splay** toward LCA
- Challenge: **concurrency**!
  - Access Lemma of splay trees no longer works: **potential function** does not „**telescope**“ anymore: a concurrently rising node may push down another rising node again

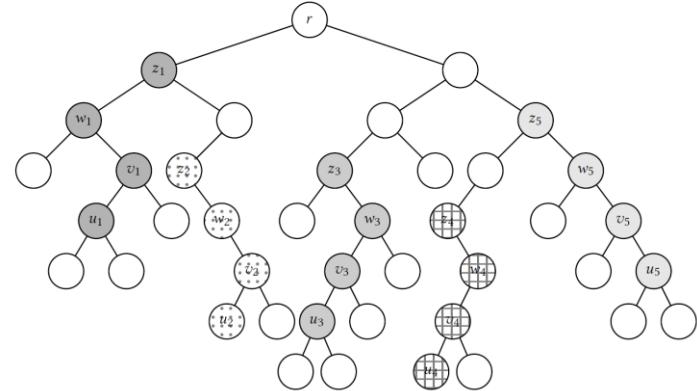


**SplayNet**

# DiSplayNet: Challenges

- DiSplayNet: Rotations (zig,zigzag,zigzag) are **concurrent**
- To avoid conflict: distributed computation of **independent clusters**
- Still challenging:

|                | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | ... | $i - 6$ | $i - 5$ | $i - 4$ | $i - 3$ | $i - 2$ | $i - 1$ | $i$ |
|----------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|---------|---------|---------|---------|---------|---------|-----|
| $\sigma_1$     | ✓   | ✓   | ✓   | ✓   | -   | -   | -   | -   | ... | -       | -       | -       | -       | -       | -       | -   |
| $\sigma_2$     | -   | X   | X   | X   | ✓   | ✓   | ✓   | -   | ... | -       | -       | -       | -       | -       | -       | -   |
| ...            | ... | ... | ... | ... | ... | ... | ... | ... | ... | ...     | ...     | ...     | ...     | ...     | ...     | ... |
| $\sigma_{m-1}$ | -   | -   | -   | -   | -   | -   | -   | -   | ... | ✓       | ✓       | -       | -       | -       | -       | -   |
| $\sigma_m$     | -   | -   | -   | -   | -   | -   | -   | -   | ... | X       | X       | ✓       | ✓       | ✓       | ✓       | -   |



|       | 1 | 2 | 3 | ... | $i$ | $i + 1$ | $i + 2$ | $i + 3$ | $i + 4$ | $i + 5$ | $i + 6$ | ... | $j$ | ... | $k$ |
|-------|---|---|---|-----|-----|---------|---------|---------|---------|---------|---------|-----|-----|-----|-----|
| $s_1$ | ✓ | ✓ | ✓ | ... | ✓   | ✓       | ✓       | ✓       | ✓       | ✓       | ...     | -   | ✓   | ... | -   |
| $d_1$ | ✓ | ✓ | ✓ | ... | ✓   | ✓       | ✓       | ✓       | ✓       | ✓       | ...     | -   | ✓   | ... | -   |
| $s_2$ | - | ✓ | ✓ | ... | ✓   | ✓       | ✓       | ✓       | -       | -       | ...     | -   | -   | ... | -   |
| $d_2$ | - | ✓ | ✓ | ... | ✓   | ✓       | X       | ✓       | -       | -       | ...     | -   | -   | ... | -   |
| $s_3$ | - | - | ✓ | ... | X   | X       | X       | ✓       | X       | X       | ...     | ✓   | ... | -   | -   |
| $d_3$ | - | - | ✓ | ... | X   | X       | X       | X       | X       | X       | ...     | ✓   | ... | -   | -   |

Sequential SplayNet: requests **one after another**

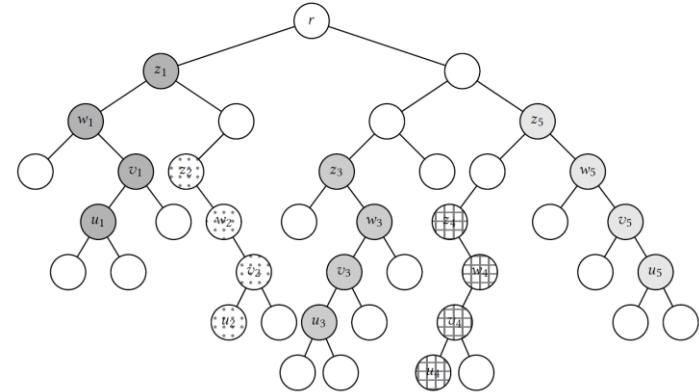
DiSplayNet: Analysis more challenging: potential function sum no longer **telescopic**. One request can “push-down” another.

# DiSplayNet: Challenges

- DiSplayNet: Rotations (zig,zigzag,zigzag) are **concurrent**
- To avoid conflict: distributed computation of **independent clusters**
- Still challenging: **Telescopic: max potential drop**

|                | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | ... | $i - 6$ | $i - 5$ | $i - 4$ | $i - 3$ | $i - 2$ | $i - 1$ | $i$ |
|----------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|---------|---------|---------|---------|---------|---------|-----|
| $\sigma_1$     | ✓   | ✓   | ✓   | ✓   |     |     |     | -   | ... | -       | -       | -       | -       | -       | -       | -   |
| $\sigma_2$     | -   | X   | X   | X   | ✓   | ✓   | ✓   | -   | ... | -       | -       | -       | -       | -       | -       | -   |
| ...            | ... | ... | ... | ... | ... | ... | ... | ... | ... | ...     | ...     | ...     | ...     | ...     | ...     | ... |
| $\sigma_{m-1}$ | -   | -   | -   | -   | -   | -   | -   | -   | ... | ✓       | ✓       |         |         |         | -       | -   |
| $\sigma_m$     | -   | -   | -   | -   | -   | -   | -   | -   | ... | X       | X       | ✓       | ✓       | ✓       | ✓       | -   |

Sequential SplayNet: requests **one after another**



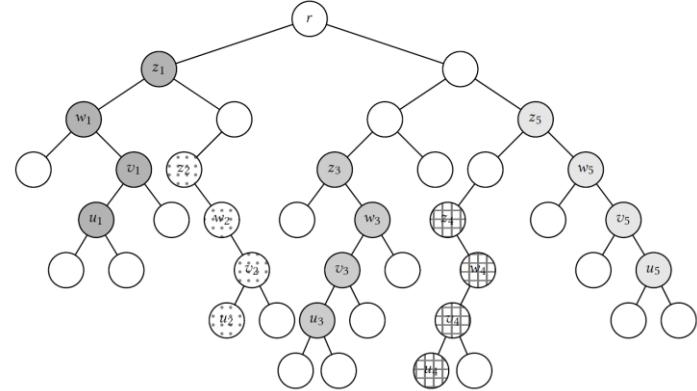
|       | 1 | 2 | 3 | ... | $i$ | $i + 1$ | $i + 2$ | $i + 3$ | $i + 4$ | $i + 5$ | $i + 6$ | ... | $j$ | ... | $k$ |
|-------|---|---|---|-----|-----|---------|---------|---------|---------|---------|---------|-----|-----|-----|-----|
| $s_1$ | ✓ | ✓ | ✓ | ... | ✓   | ✓       | ✓       | ✓       | ✓       | ✓       | ...     | -   | ✓   | ... | -   |
| $d_1$ | ✓ | ✓ | ✓ | ... | ✓   | ✓       | ✓       | ✓       | ✓       | ✓       | ...     | -   | ✓   | ... | -   |
| $s_2$ | - | ✓ | ✓ | ... | ✓   | ✓       | ✓       | ✓       | -       | -       | ...     | -   | -   | ... | -   |
| $d_2$ | - | ✓ | ✓ | ... | ✓   | ✓       | X       | ✓       | -       | -       | ...     | -   | -   | ... | -   |
| $s_3$ | - | - | ✓ | ... | X   | X       | X       | ✓       | X       | X       | ...     | ✓   | ... | -   | -   |
| $d_3$ | - | - | ✓ | ... | X   | X       | X       | X       | X       | X       | ...     | ✓   | ... | -   | -   |

DiSplayNet: Analysis more challenging: potential function sum no longer **telescopic**. One request can “push-down” another.

# DiSplayNet: Challenges

- DiSplayNet: Rotations (zig,zigzag,zigzag) are **concurrent**
- To avoid conflict: distributed computation of **independent clusters**
- Still challenging: **Telescopic: max potential drop**

|                | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | ... | $i - 6$ | $i - 5$ | $i - 4$ | $i - 3$ | $i - 2$ | $i - 1$ | $i$ |
|----------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|---------|---------|---------|---------|---------|---------|-----|
| $\sigma_1$     | ✓   | ✓   | ✓   | ✓   |     |     |     |     | ... | -       | -       | -       | -       | -       | -       | -   |
| $\sigma_2$     | -   | X   | X   | X   | ✓   | ✓   | ✓   | -   | ... | -       | -       | -       | -       | -       | -       | -   |
| ...            | ... | ... | ... | ... | ... | ... | ... | ... | ... | ...     | ...     | ...     | ...     | ...     | ...     | ... |
| $\sigma_{m-1}$ | -   | -   | -   | -   | -   | -   | -   | -   | ... | ✓       | ✓       |         |         |         |         | -   |
| $\sigma_m$     | -   | -   | -   | -   | -   | -   | -   | -   | ... | X       | X       | ✓       | ✓       | ✓       | ✓       | -   |



---

## Remark 3: Accounting for Congestion

---

# A Tradeoff?!



Short routes:  
congestion

VS

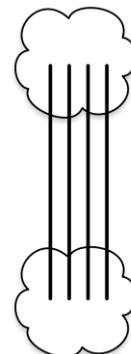


Low congestion:  
long routes

# A Tradeoff?!



Short routes:  
congestion



Or both?

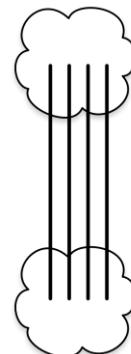


Low congestion:  
long routes

# A Tradeoff?!



Short routes:  
congestion



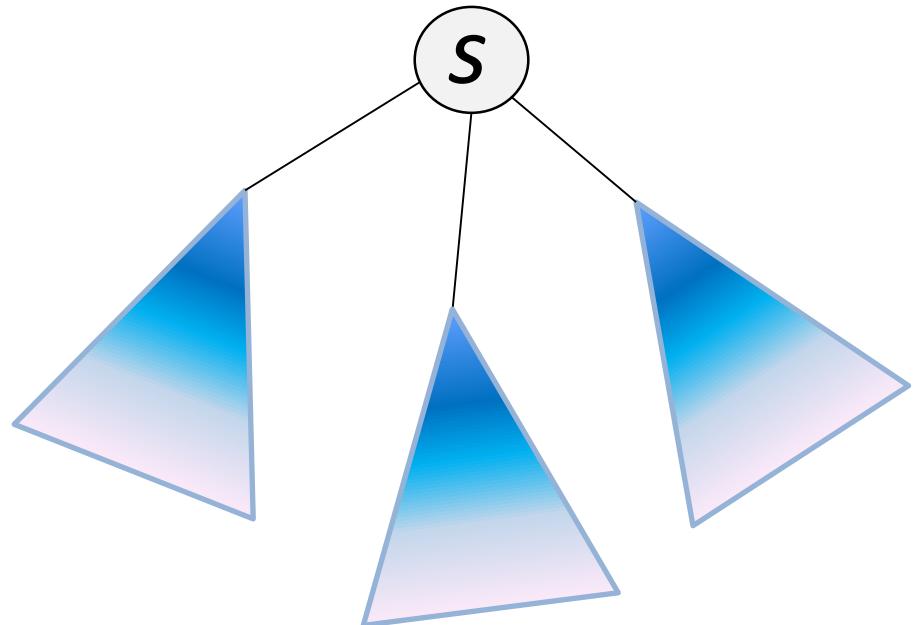
Or both?



Low congestion:  
long routes

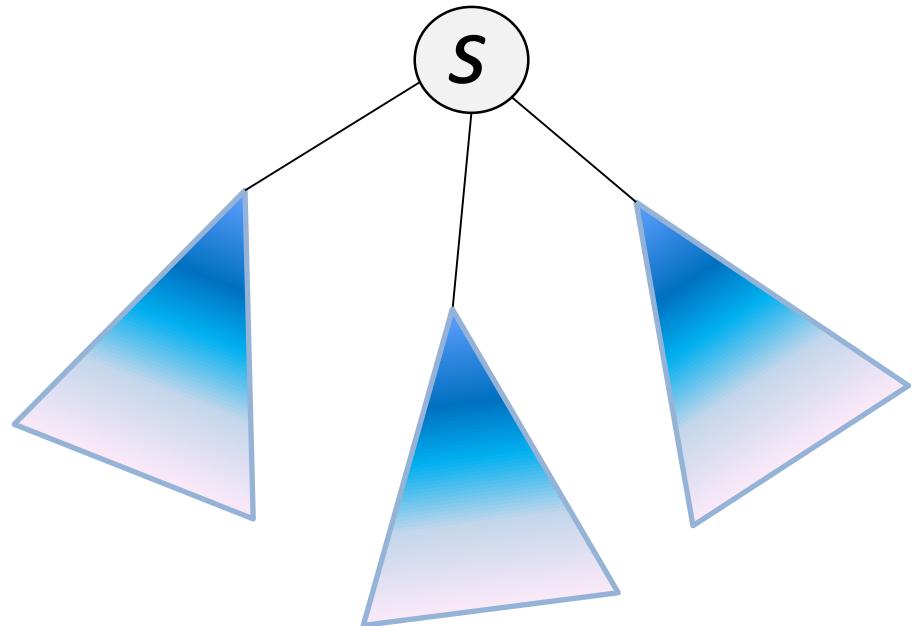
# Ego-Tree++!

- Idea: place destination nodes greedily across subtrees s.t.  
*congestion balanced*
- ... while preserving distance
- Trees can have different sizes but *similar mass!*
- Bicriteria guarantee



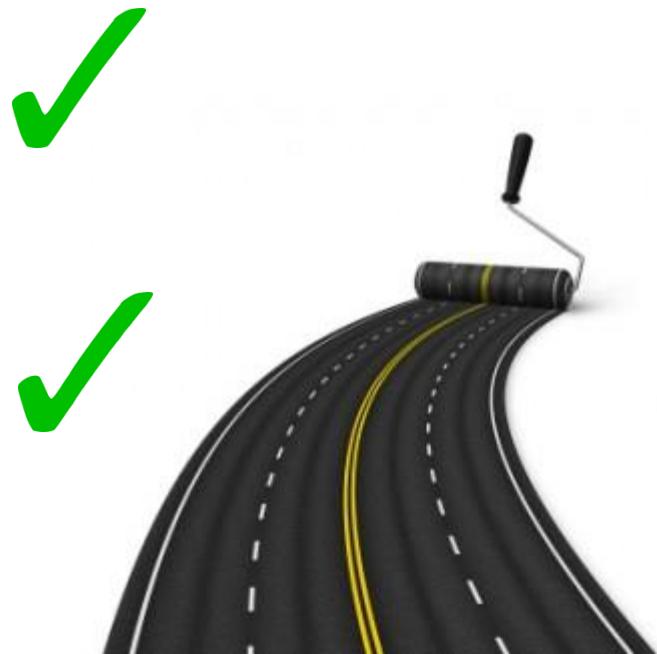
# Ego-Tree++!

- Idea: place destination nodes greedily across subtrees s.t.  
*congestion balanced*
- ... while preserving distance
- Trees can have different sizes but *similar mass!*
- Bicriteria guarantee



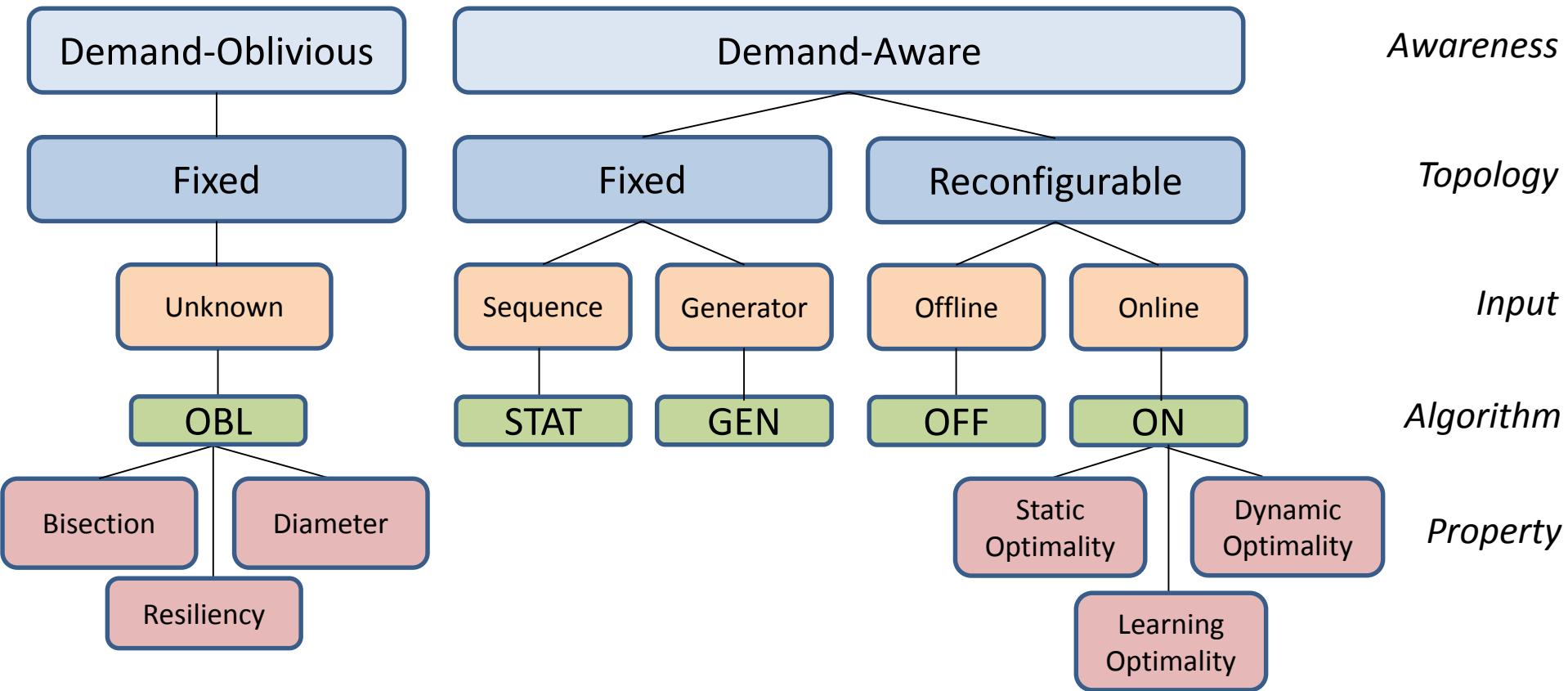
# Roadmap

- Entropy: A metric for demand-aware networks?
  - Intuition
  - A lower bound
  - Algorithms achieving entropy bounds
- From static to dynamic demand-aware networks
  - Empirical motivation
  - A connection to self-adjusting datastructures



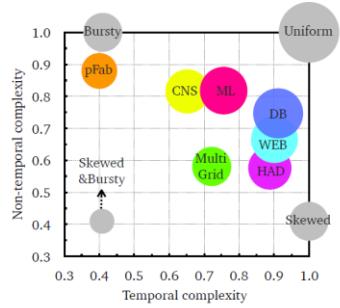
# Uncharted Landscape!

Toward Demand-Aware Networking: A Theory for  
Self-Adjusting Networks. SIGCOMM CCR, 2018.

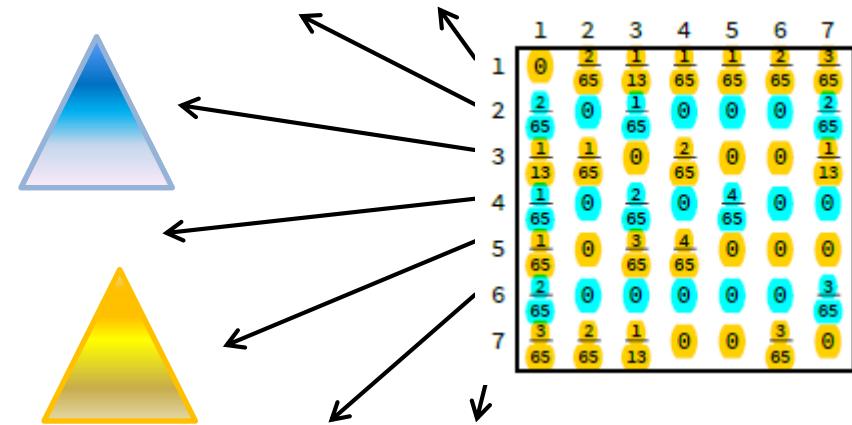


# Big Open Questions

- Cross-layer aspects
- Metrics: just the beginning!
- We need more data
- Unifying theory
- How to convince operators?



# Thank you! Questions?



# Further Reading

## Demand-Aware and Self-Adjusting Networks

[Survey of Reconfigurable Data Center Networks: Enablers, Algorithms, Complexity](#)

Klaus-Tycho Foerster and Stefan Schmid.

**SIGACT News**, June 2019.

[Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks \(Editorial\)](#)

Chen Avin and Stefan Schmid.

ACM SIGCOMM Computer Communication Review (**CCR**), October 2018.

[Demand-Aware Network Design with Minimal Congestion and Route Lengths](#)

Chen Avin, Kaushik Mondal, and Stefan Schmid.

38th IEEE Conference on Computer Communications (**INFOCOM**), Paris, France, April 2019.

Documents: paper [pdf](#), bibtex [bib](#)

[Distributed Self-Adjusting Tree Networks](#)

Bruna Peres, Otavio Augusto de Oliveira Souza, Olga Goussevskaia, Chen Avin, and Stefan Schmid.

38th IEEE Conference on Computer Communications (**INFOCOM**), Paris, France, April 2019.

[Efficient Non-Segregated Routing for Reconfigurable Demand-Aware Networks](#)

Thomas Fenz, Klaus-Tycho Foerster, Stefan Schmid, and Anaïs Villedieu.

**IFIP Networking**, Warsaw, Poland, May 2019.

[DaRTree: Deadline-Aware Multicast Transfers in Reconfigurable Wide-Area Networks](#)

Long Luo, Klaus-Tycho Foerster, Stefan Schmid, and Hongfang Yu.

IEEE/ACM International Symposium on Quality of Service (**IWQoS**), Phoenix, Arizona, USA, June 2019.

[Demand-Aware Network Designs of Bounded Degree](#)

Chen Avin, Kaushik Mondal, and Stefan Schmid.

31st International Symposium on Distributed Computing (**DISC**), Vienna, Austria, October 2017.

[SplayNet: Towards Locally Self-Adjusting Networks](#)

Stefan Schmid, Chen Avin, Christian Scheideler, Michael Borokhovich, Bernhard Haeupler, and Zvi Lotker.

IEEE/ACM Transactions on Networking (**TON**), Volume 24, Issue 3, 2016. Early version: IEEE **IPDPS** 2013.

[Characterizing the Algorithmic Complexity of Reconfigurable Data Center Architectures](#)

Klaus-Tycho Foerster, Monia Ghobadi, and Stefan Schmid.

ACM/IEEE Symposium on Architectures for Networking and Communications Systems (**ANCS**), Ithaca, New York, USA, July 2018.